

高等教育“十三五”规划教材

单片机原理及应用案例教程

主 编 禹定臣 李白燕

副主编 张 健 李 平 魏迎军

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是根据教育部应用型科技大学的教学要求和最新大纲编写而成的。全书以 MCS-51 系列单片机为例,通过丰富的应用实例,采用理论和实践相结合的方法,由浅入深地介绍了 51 系列单片机的结构及工作原理、内部硬件资源及单片机的系统扩展、A/D、D/A、常用接口设计及常用的编程语言(汇编语言与 C51)和开发工具(Proteus、Keil C)的使用等内容,并介绍了单片机应用系统的设计、开发与调试过程。

本书注重学生能力的培养,采用案例教学,融“教、学、练”于一体,案例中将 Proteus 和 keil C 相结合,实践性和可操作性强。编程以 C51 为主,兼顾汇编语言程序设计。最后给出了详细的单片机实验指导和课程设计实例,供实践教学参考。

本书可作为高等院校电子信息工程、通信工程、电子科学与技术、计算机、自动化、机电一体化等相关专业的本、专科教材和参考书,也可供从事单片机应用开发的工程技术人员及其他工程技术人员参考,同时还可以作为全国大学生电子设计竞赛的培训教材。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

单片机原理及应用案例教程 / 禹定臣, 李白燕主编. —北京: 电子工业出版社, 2017.3
ISBN 978-7-121-30521-4

I. ①单… II. ①禹… ②李… III. ①单片微型计算机—高等学校—教材 IV. ①TP368.1

中国版本图书馆 CIP 数据核字(2016)第 289653 号

策划编辑: 祁玉芹

责任编辑: 张瑞喜

印 刷: 中国电影出版社印刷厂

装 订: 中国电影出版社印刷厂

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 20.5 字数: 499 千字

版 次: 2017 年 3 月第 1 版

印 次: 2017 年 3 月第 1 次印刷

定 价: 45.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式:(010) 68253127。

前言

P R E F A C E

单片机原理及应用是工科类院校开设的一门面向应用、具有很强的实践性与综合性的课程，为了培养学生应用单片机技术进行电子产品的软硬件设计和制作能力，积累开发经验，通过对传统的教学模式改革、案例教学，使初学者理解单片机的工作过程和应用系统的开发流程，使用 Labcenter 公司的 Proteus 仿真软件创建实验环境，进行软、硬件仿真，做到了融“教、学、练”于一体，“边学理论，边实践”。

本书采用案例教学，内容安排合理，定位准确，实用性强，注重实践能力的培养，满足应用型科技大学的教学目标、培养方向和办学特色的需要，以能力培养为目标、以工作过程为导向，用案例贯穿知识，用任务驱动教学，内容精炼，注重实用。

全书共分 10 章。

第 1 章介绍了 51 单片机的系统结构及引脚功能；第 2 章介绍了 51 单片机程序设计基础；第 3 章介绍了单片机中断系统、定时/计数器、串行口；第 4 章介绍了单片机的系统扩展；第 5 章和第 6 章分别介绍了 A/D 转换、D/A 转换及常用接口的设计；第 7 章介绍了单片机应用系统的设计、开发与调试方法；第 8 章介绍了单片机常用开发工具；第 9 章和第 10 章分别给出了实验指导和课程设计实例。

本书由禹定臣、李白燕担任主编，张健、李平、魏迎军担任副主编。由电子信息工程、通信工程专业教学一线教师合作编写完成。第 1 章～第 2 章由李平编写，第 3 章～第 4 章由李白燕编写，第 5 章～第 7 章由张健编写，第 9 章～第 10 章及实例由禹定臣编写，第 8 章、附录、习题由魏迎军编写。全书由禹定臣教授定稿，耿红琴教授主审。在编写过程中参阅借鉴了一些相关教材和文献，在此向有关编者表示感谢。

由于编写时间仓促，书中难免有疏漏和不妥之处，欢迎读者批评指正，以便再版时及时修正。

编者
2017.1

目 录

C O N T E N T S

第 1 章 MCS-51 单片机硬件结构	1
1.1 知识结构	1
1.1.1 单片机内部结构	1
1.1.2 引脚功能	15
1.2 学习实例	17
实例一 LED 灯闪烁	17
实例二 LED 流水灯	19
实例三 转向灯	21
本章小结	22
习题一	22
第 2 章 单片机汇编语言与 C 语言程序设计基础	25
2.1 知识结构	25
2.1.1 汇编语言程序设计	25
2.1.2 C51 程序设计	49
2.2 学习实例	66
实例一 用 P1 口、P2 口分别显示二进制加、减法结果	66
实例二 用 P2 口实现左右跑马灯效果	68
实例三 用查表法实现 P2 口接的 8 只 LED 灯花样显示	71
本章小结	73
习题二	73
第 3 章 单片机中断系统、定时器/计数器及串行口	77
3.1 知识结构	77
3.1.1 中断系统	77
3.1.2 定时器/计数器	82

3.1.3 串行口	86
3.2 学习实例	92
实例一 用 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 对按键进行计数并显示计数结果	92
实例二 用 T0 工作在方式 1 时控制播放一首歌曲	94
实例三 用 T0 工作在方式 1 时控制 LED 灯的闪烁时间间隔	96
实例四 用 T0 工作在方式 2 时对脉冲进行计数并显示计数结果	98
实例五 用串行口工作在方式 0 时扩展输出接口	99
实例六 用串行口工作在方式 1 时实现双机通信	101
实例七 单片机向 PC 机发送数据	102
实例八 单片机接收 PC 机发送的数据	104
本章小结	106
习题三	106
第 4 章 51 单片机系统扩展	109
4.1 知识结构	109
4.1.1 单片机系统总线及系统扩展方法	109
4.1.2 单片机存储器的扩展	112
4.1.3 并行 I/O 口扩展	117
4.2 学习实例	131
实例一 用 62256 扩展 32KB 的外部 RAM	131
实例二 用 27256 扩展 32KB 的外部 ROM	132
实例三 用 AT24C02 扩展 EEPROM	133
实例四 用 74LS273、74LS241 扩展 I/O 接口	137
实例五 用 8255 芯片扩展键盘/显示接口	139
实例六 用 8155 芯片扩展显示接口	142
实验七 用 74ls165、74ls164 扩展键盘/显示接口	144
本章小结	145
习题四	145
第 5 章 DAC 和 ADC 接口	148
5.1 知识结构	148
5.1.1 A/D 转换器件	148
5.1.2 D/A 转换器件	150
5.2 学习实例	152
实例一 基于 ADC0809 的 5V 直流电压表设计	152
实例二 用 DAC0832 设计简易信号发生器	154
本章小结	157
习题五	157

第 6 章 键盘与显示接口设计	160
6.1 知识结构	160
6.1.1 键盘接口设计	160
6.1.2 LED 显示接口设计	165
6.1.3 LCD 显示接口设计	168
6.2 学习实例	173
实例一 用 LED 数码管循环显示 0~9	173
实例二 用 LED 数码管动态显示“HELLO”	175
实例三 数码时钟设计	176
实例四 独立式键盘控制步进电动机正、反转	180
实例五 矩阵式键盘按键值的数码管显示	183
实例六 矩阵式键盘按键值的 LCD 显示	185
实例七 用 1602LCD 显示“HUANG HUAI UNIVERSITY”	187
实例八 用 12864LCD 显示汉字	190
本章小结	193
习题六	194
第 7 章 单片机应用系统设计与调试简介	195
7.1 知识结构	195
7.1.1 单片机应用系统的设计步骤	195
7.1.2 应用系统的硬件设计	196
7.1.3 应用系统的软件设计	197
7.1.4 单片机应用系统的开发与调试	197
7.1.5 单片机应用系统的可靠性与抗干扰性设计	199
7.2 学习实例	204
实例一 基于 DS1302 的日历时钟设计	204
实例二 基于 ADC0832 和 LCD1602 的数字电压表设计	212
本章小结	215
习题七	215
第 8 章 常用开发仿真软件 Keil c 和 Proteus 简介	216
8.1 知识结构	216
8.1.1 Keil C 编译器使用简介	216
8.1.2 Proteus 仿真软件使用简介	221
8.2 学习实例	225
实例一 通过 P1.0 输出周期为 20ms 的方波信号	225

实例二 计单个按键次数并显示	227
本章小结	228
习题三	229
第 9 章 单片机实验指导	230
实验一 P1 口实验	230
实验二 交通灯控制实验	234
实验三 简单 I/O 口扩展实验	238
实验四 外部中断实验	243
实验五 定时器实验	247
实验六 8255A 可编程并行接口实验	251
实验七 数码显示实验	253
实验八 液晶显示屏 1602 显示实验	256
实验九 串/并转换实验	260
实验十 A/D 转换实验	263
第 10 章 单片机课程设计实例	267
实例一 基于单片机的简易计算器设计	267
实例二 基于单片机的数字电压表设计	272
实例三 基于单片机的电子日历设计	276
实例四 基于单片机的具备温度显示的数字时钟设计	284
实例五 基于单片机的具备转速显示功能的直流电动机控制系统设计	293
实例六 基于单片机的红外遥控器控制继电器的设计	297
附录	304
附录 A MCS-51 系列单片机指令表	304
附录 B Protreus 的常用元器件	309
附录 C C51 常用库函数	314
参考文献	319

第 1 章 MCS-51 单片机硬件结构

学习目标

掌握 MCS-51 单片机的内部结构和引脚功能，并行 I/O 口的功能和使用方法，存储器空间分布，常用的特殊功能寄存器，单片机典型时钟电路，典型复位电路及复位对单片机各部件的影响。

重点难点

- (1) MCS-51 单片机的内部结构和芯片引脚功能。
- (2) I/O 口的工作过程和片内 RAM 寻址及常用 SFR 的作用。
- (3) 典型复位电路及复位对单片机各部件的影响。
- (4) I/O 口工作过程、片内 RAM、复位电路。

1.1 知识结构

1.1.1 单片机内部结构

单片机是集成在一片集成芯片上的微型计算机，由 CPU、存储器（ROM、RAM）、串行接口、并行接口、定时器/计数器、中断系统、振荡器和时钟电路等组成，各部分之间通过系统总线相连。MCS-51 单片机的基本结构如图 1.1 所示。

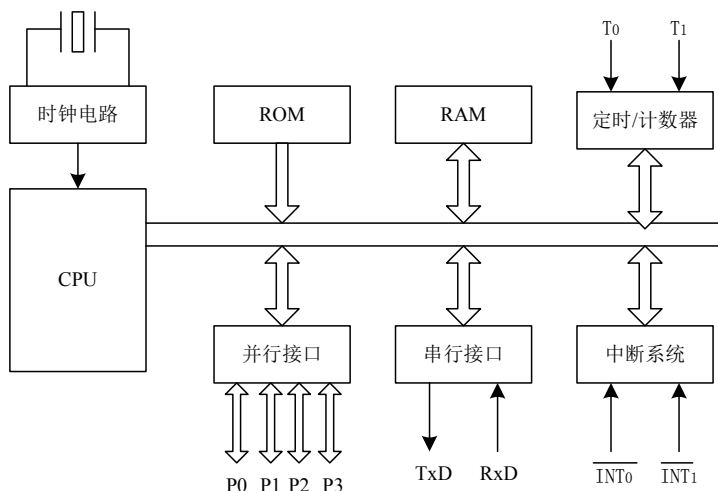


图 1.1 MCS-51 单片机基本结构

1. CPU

MCS-51 单片机 8 位 CPU 内部结构如图 1.2 所示。内部主要由算术逻辑单元 ALU (Arithmetic Logic Unit)、累加器 A (8 位)、寄存器 B (8 位)、程序计数器 PC、程序状态字 PSW (8 位)、指令寄存器 IR (8 位地址)、指令译码器 ID、地址寄存器 AR、数据寄存器 DR 和定时与控制电路等部件组成。

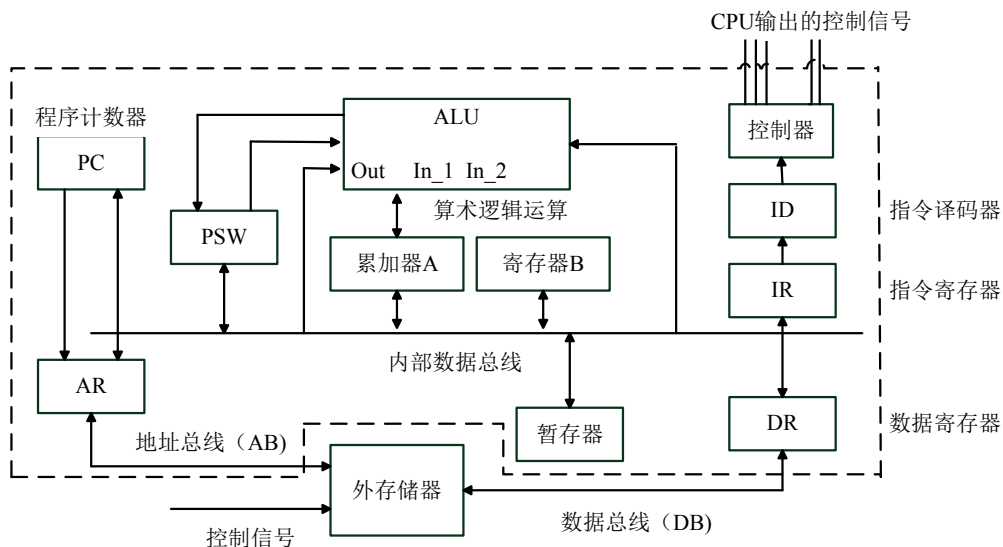


图 1.2 51 单片机 CPU 内部结构方框图

(1) 运算器 (ALU)

主要功能是进行算术和逻辑运算，可对半字节 (一个字节是 8 位，半个字节就是 4 位) 和单字节数据进行操作，具有位处理功能 (即布尔处理器)。

(2) 累加器 A (Accumulator)

累加器 A 简称为 ACC 或 A 寄存器，为 8 位寄存器，是 CPU 中最繁忙、使用频率最高的一个特殊功能寄存器，它既可用于存放操作数，也可用来存放运算的中间结果。

(3) 寄存器 B

寄存器 B 是专门为乘法和除法运算设置的 8 位寄存器。在乘法中，ALU 的两个输入分别是 A 寄存器和 B 寄存器，乘积在 AB 寄存器对中，A 存低 8 位，B 存高 8 位；在除法中，被除数取自 A，除数取自 B，商在 A 中，余数在 B 中。其他情况下，B 寄存器可以作为一个普通的寄存器使用。

(4) 程序计数器 PC

PC 的作用是指明即将执行的下一条指令的地址，共 16 位，可对 64K ROM 直接寻址，PC 低 8 位经 P0 口输出，高 8 位经 P2 口输出。程序执行到什么地方，程序计数器 PC 就指到那里，程序计数器 PC 具有自动加 1 的功能，即从存储器中读出一个字节的指令码后，PC 自动加 1 (指向下一个存储单元)。

PC 没有地址，是不可寻址的，因此用户无法对它进行读写，但可以通过转移、调用、

返回等特殊指令改变其内容，以实现程序的转移。因地址不在 SFR 之内，一般不用作专用寄存器。

(5) 程序状态字 PSW

PSW (Program Status Words) 是一个 8 位的专用寄存器，包含程序运行的状态信息，用于记录运算过程中的状态，其中有些状态位是根据程序执行的结果，由硬件自动设置的，而有些状态位则使用软件方法设定。PSW 的状态位可以用专门指令进行测试，也可以用指令读出。一些条件转移指令将根据 PSW 有些位的状态，进行程序转移。PSW 的各位定义如下：

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	—	P

除 PSW.1 位保留未用外，对其余各位的定义及使用如下：

CY (PSW.7) ——进位或借位标志位。CY 也可以写为 C，CPU 进行加或减运算时，用于存放算术运算的进位或借位标志。如果运算结果最高位有进位或借位时，CY 由硬件置 1，否则清 0；在位操作中，作为累加器使用。

AC (PSW.6) ——辅助进位标志位。在进行 BCD 码加减运算中，当有低 4 位向高 4 位进位或借位时，AC 由硬件置 1，否则 AC 位被清 0。

F0 (PSW.5) ——用户标志位。是一个供用户定义的标志位，要利用软件方法置位或复位，用户编程时通过设置 F0 以控制程序的走向。

RS1 和 RS0 (PSW.4, PSW.3) ——工作寄存器组选择位。用于选择 CPU 当前使用的工作寄存器组。工作寄存器共有四组，其对应关系如表 1.1 所示。

表 1.1 RS1、RS0 对工作寄存器的选择

RS1	RS0	寄存器组	片内 RAM 地址
0	0	第 0 组	00H~07H
0	1	第 1 组	08H~0FH
1	0	第 2 组	10H~17H
1	1	第 3 组	18H~1FH

这两个选择位的状态是由软件设置的，被选中的工作寄存器组即为当前工作寄存器组。当单片机上电或复位后，RS1RS0=00。

OV (PSW.2) ——溢出标志位。在带符号数加减运算中，OV=1 表示加减运算超出了累加器 A 所能表示的符号数的有效范围 (-128~+127)，即产生了溢出，因此运算结果是错误的；否则，OV=0 表示运算正确，即无溢出产生。

在乘法运算中，OV=1 表示乘积超过 255，即乘积分别在 B 与 A 中；否则，OV=0，表示乘积只在 A 中。

在除法运算中，OV=1 表示除数为 0，除法不能进行；否则，OV=0，除数不为 0，除法可正常进行。

P (PSW.0) ——奇偶标志位。表明累加器 A 中包含 1 的个数的奇偶性，如果 A 中有奇数个 1，则 P 置 1，否则置 0。改变累加器 A 中内容的指令均会影响 P 标志位。

(6) 指令寄存器 IR

指令寄存器的作用就是用来存放即将执行的指令代码。

CPU 执行指令的过程：首先由程序存储器（ROM）中读取指令代码送入指令寄存器，经译码器译码后再由定时与控制电路发出相应的控制信号，从而完成指令的功能。

(7) 指令译码器 ID

用于对送入指令寄存器中的指令进行译码，所谓译码就是把指令转变成执行此指令所需要的电信号。当指令送入译码器后，由译码器对该指令进行译码，根据译码器输出的信号，CPU 控制电路定时地产生执行该指令所需的各种控制信号，使单片机正确地执行程序所需要的各种操作。

指令的执行分为三个阶段：取指令、分析指令和执行指令。

具体步骤：首先，进行程序存储器读操作，也就是根据程序计数器 PC 给出的地址从程序存储器中取出指令，送指令寄存器 IR，IR 输出到指令译码器 ID，然后，指令译码器对该指令进行译码，控制逻辑产生一系列控制信号，送到单片机的各部件中，控制执行这一指令，执行的结果影响程序状态标志寄存器 PSW 的内容。指令寄存、译码控制逻辑电路图如图 1.3 所示。

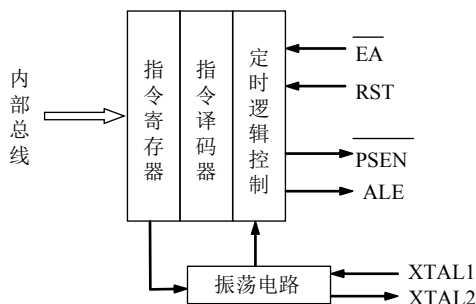


图 1.3 指令寄存、译码控制逻辑电路图

(8) 地址寄存器 AR

AR 的作用是用来存放将要寻址的外部存储器单元的地址信息，指令码所在存储单元的地址编码，由程序计数器 PC 产生，而指令中操作数所在的存储单元地址码，由指令的操作数给定。从图 1.2 中可以看到，地址寄存器 AR 通过地址总线 AB 与外部存储器相连。

(9) 数据寄存器 DR

用于存放写入外部存储器或 I/O 端口的数据信息。数据寄存器对输出数据具有锁存功能，数据寄存器与外部数据总线 DB 直接相连。

(10) 定时与控制电路

定时与控制电路的功能是产生 CPU 的操作时序，它是单片机的核心，控制各种操作的时序。

2. 存储器结构电路

MCS-51 单片机采用的是哈佛（Harvard）结构，即程序存储器空间和数据存储器空间是各自独立的，两种存储器各有自己的寻址方式和寻址空间。存储器有 4 个物理上相互独立的存储器空间：片内、片外程序存储器和片内、片外数据存储器。基本结构电路如图 1.4

所示。

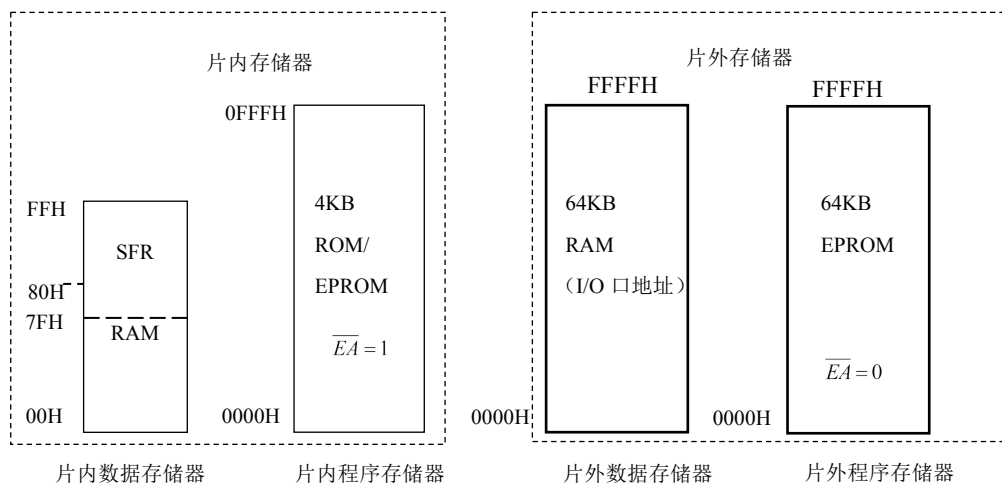


图 1.4 MCS-51 单片机存储器的结构电路

（1）程序存储器

程序是指指挥单片机动作的一系列命令，单片机能直接认识并执行的命令是一串由“0”和“1”代码组成的机器指令。在执行程序之前，必须把编好的程序和所需表格常数存入程序存储器中，单片机中用于存储程序的物理器件就是程序存储器。程序存储器以 16 位的程序计数器 PC 为地址指针，可寻址空间为 64 KB。

8051 片内有 4 KB 的 ROM，8751 片内有 4 KB 的 EPROM，8031 片内无程序存储器。8051 和 8751 单片机程序存储器空间有片内和片外两部分组成，片内程序存储器由 0000H~0FFFH 最低 4KB 组成，片外扩展的程序存储器由 1000H~FFFFH（60KB）。

CPU 访问片内和片外程序存储器，可由 \overline{EA} 引脚所接的电平来确定。当 \overline{EA} 引脚接高电平时，CPU 将首先从片内程序存储器中取指令，当指令地址（即 PC 值）超过 0FFFH 时，会自动转向片外程序存储器取指令。当 \overline{EA} 引脚接地或低电平时，CPU 只能访问片外程序存储器。由于 8031 单片机无片内程序存储器，应将 \overline{EA} 引脚固定接地或低电平。

单片机的程序存储器是只读的，CPU 可以通过 MOV C 指令遍访 64KB 的程序存储器空间，但没有任何其他指令可以用来更改程序存储区的内容，即向程序存储器执行写入操作。

MCS-51 的程序存储器中有些单元具有特殊功能，使用时应注意。

其中一组特殊单元是 0000H~0002H。系统复位后，(PC)=0000H，单片机从 0000H 单元开始取指令执行程序。如果程序不从 0000H 单元开始，应在这三个单元中存放一条无条件转移指令，以便直接转去执行指定的程序。

还有一组特殊单元是 0003H~002AH，共 40 个单元。这 40 个单元被均匀地分为 5 段，作为 5 个中断源的中断服务程序区。

实际应用中，用户可根据需要扩展程序存储器的容量，而其地址空间原则上也可由用户任意安排。但程序存储器中有 6 个单元被固定用于复位和中断服务程序的入口地址，见表 1.2。

表 1.2 单片机复位和中断源的中断入口地址

程序入口地址	用 途
0000H	复位后初始化引导程序地址
0003H	外部中断 0 ($\overline{INT0}$) 中断服务程序入口
000BH	定时器 0 (T0) 溢出中断服务程序入口
0013H	外部中断 1 ($\overline{INT1}$) 中断服务程序入口
001BH	定时器 1 (T1) 溢出中断服务程序入口
0023H	串行口中断服务程序入口

单片机复位后程序计数器 PC 的内容为 0000H，程序存储器 0000H 单元是系统程序的起始地址，一般在该单元存放一条无条件转移指令来改变 PC 值，将 PC 指向用户设计的程序（主程序或初始化程序）的入口地址，程序将从转移后的地址开始存放。

当中断响应后，系统能根据中断源的种类自动转到相应中断源的服务程序入口地址去执行程序。但由于 2 个中断源入口地址间隔仅有 8 个单元，这 8 个存储单元难以存储一个完整的中断服务程序，所以在中断入口地址单元一般也存放一条无条件转移指令转向中断服务程序的实际入口地址。

(2) 数据存储器

① 内部数据存储器。

MCS-51 单片机的内部 RAM 共有 256 个单元，按其功能划分为两部分：低 128 单元（单元地址 00H~7FH）和高 128 单元（单元地址 80H~FFH）。低 128 单元的配置如表 1.3 所列。

表 1.3 片内 RAM 低 128 单元配置

30H~7FH	数据缓冲区、堆栈区
20H~2FH	位寻址区（00H~7FH）
18H~1FH	工作寄存器 3 区（R7~R0）
10H~17H	工作寄存器 2 区（R7~R0）
08H~0FH	工作寄存器 1 区（R7~R0）
00H~07H	工作寄存器 0 区（R7~R0）

低 128 单元是单片机的真正 RAM 存储器，按其用途划分为以下 3 个区域。

● 工作寄存器区（00H~1FH）。

在片内 RAM 的最低端共有四组工作寄存器，每组 8 个存储单元（各为 8 位），各组都以 R0~R7 作寄存器编号，常用于存放操作数及中间结果等，也称之为通用寄存器或工作寄存器。四组工作寄存器占据内部 RAM 的 00H~1FH 单元。

在某一时刻，CPU 只能使用其中的一组寄存器，正在使用的那组寄存器称为当前寄存器组。没选用的工作寄存器组所对应的单元可以作为一般的通用寄存器使用。用户可以通过指令改变程序状态字寄存器 PSW 中的 PSW.3 (RS0) 和 PSW.4 (RS1) 来选择哪一组寄存器作为当前的工作寄存器组。通用寄存器为 CPU 提供了就近数据存储的便利，有利于提高单片机的运算速度。使用通用寄存器还能提高程序编制的灵活性，在单片机的应用编程中应充分利用这些寄存器，以简化程序设计，提高程序运行速度。

● 位寻址区（20H~2FH）。

内部 RAM 的 20H~2FH 单元，既可作为一般 RAM 单元使用，进行字节操作，也可以进行 128 位的位操作，因此把该区称之为位寻址区，位地址为 00H~7FH。MCS-51 具有

布尔处理机功能，位寻址区就是布尔处理机的存储空间。表 1.4 所示为位寻址区的位地址。

表 1.4 内部 RAM 位寻址区的位地址

单元地址	位地址							
	MSB							LSB
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

- 用户 RAM 区（30H~7FH）。

内部 RAM 的 128 单元中，通用寄存器占去 32 个单元，位地址区占去 16 个单元，剩下 80 个单元，其单元地址为 30H~7FH，是供用户使用的一般 RAM 区，可以作为数据缓冲区分以及堆栈区使用。

在 51 单片机内部 RAM 的高 128 单元中离散地分布有 21 个特殊功能寄存器，单元地址范围为 80H~FFH。这些寄存器的功能已作专门规定，故而称之为专用寄存器（Special Function Register），也称为特殊功能寄存器。这里作部分介绍，其中程序计数器 PC、累加器 A、寄存器 B 和程序状态字 PSW 前面已介绍，现把堆栈指针寄存器 SP 和数据指针 DPTR 作简单介绍。

- 堆栈指示器 SP（Stack Pointer）。

堆栈是一种后进先出（LIFO）的线性表，使用单片机内部 RAM 单元存储一些需要回避的数值数据或地址数据。堆栈好像堆放货物的仓库一样，存取数据时采用“后进先出”（即“先进后出”）的原则。堆栈指针 SP 是用来存放当前堆栈栈顶指向的存储单元地址的一个 8 位特殊功能寄存器，地址是 81H。

堆栈主要是为子程序调用和中断操作而设立的，常用的功能有两个：保护断点和保护现场。堆栈只有两种操作：入栈和出栈。不论数据是入栈还是出栈，都是对栈顶单元（SP 指向的单元）进行操作的。51 单片机堆栈是向上生成的，入栈时 SP 内容是增加的，出栈时 SP 的内容是减少的。堆栈区域的大小可用软件对 SP 重新定义初值来改变，但堆栈深度以不超过片内 RAM 空间为限。系统复位后，SP 的值为 07H，若不重新定义，则以 07H 单元为栈底，入栈的内容从地址为 08H 单元开始存放。

- 数据指针 DPTR（Data pointer）。

数据指针为 16 位寄存器，其高位字节寄存器用 DPH 表示，低位字节寄存器用 DPL 表示。编程时，DPTR 既可以按 16 位寄存器使用，也可以按两个 8 位寄存器分开使用，DPH

是 DPTR 高位字节，DPL 是 DPTR 低位字节。DPTR 通常在访问外部数据存储器时作地址指针使用。由于外部数据存储器的寻址范围为 64 KB，故把 DPTR 设计为 16 位。

对专用寄存器的字节寻址问题作如下说明：

21 个可字节寻址的专用寄存器是不连续地分散在内部 RAM 高 128 单元之中，尽管还余有许多空闲地址，但用户并不能使用；

程序计数器 PC 不占据 RAM 单元，它在物理上是独立的，因此是不可寻址的寄存器，专用寄存器只能使用直接寻址方式，书写时既可使用寄存器符号，也可使用寄存器地址。

- 专用寄存器位地址空间。

MCS-51 系列单片机有 21 个可寻址的专用寄存器，其中有 11 个专用寄存器是可以位寻址的。下面把各寄存器的字节地址及位地址列于表 1.5 中。

表 1.5 MCS-51 专用寄存器地址表

SFR	MSB		位地址/位定义				LSB		字节地址
B	F7	F6	F5	F4	F3	F2	F1	F0	F0H
ACC	E7	E6	E5	E4	E3	E2	E1	E0	E0H
PSW	D7	D6	D5	D4	D3	D2	D1	D0	D0H
	CY	AC	F0	RS1	RS0	OV	F1	P	
IP	BF	BE	BD	BC	BB	BA	B9	B8	B8H
	/	/	/	PS	PT1	PX1	PT0	PX0	
P3	B7	B6	B5	B4	B3	B2	B1	B0	B0H
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	
IE	AF	AE	AD	AC	AB	AA	A9	A8	A8H
	EA	/	/	ES	ET1	EX1	ET0	EX0	
P2	A7	A6	A5	A4	A3	A2	A1	A0	A0H
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
SBUF									(99H)
SCON	9F	9E	9D	9C	9B	9A	99	98	98H
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
P1	97	96	95	94	93	92	91	90	90H
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
TH1									(8DH)
TH0									(8CH)
TL1									(8BH)
TL0									(8AH)
TMOD	GATE	C/T	M1	M0	GAT	C/T	M1	M0	(89H)
TCON	8F	8E	8D	8C	8B	8A	89	88	88H
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
PCON	SMOD	/	/	/	/	/	/	/	(87H)
DPH									(83H)
DPL									(82H)
SP									(81H)
P0	87	86	85	84	83	82	81	80	80H
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	

表中凡字节地址不带括号的寄存器都是可进行位寻址的寄存器，而带括号的是不可位寻址的寄存器。全部专用寄存器可寻址的位共 83 位，这些位都具有专门的定义和用途。加

上位寻址区的 128 位，在 MCS-51 的内部 RAM 中共有 $128+83=211$ 个可寻址位。

② 外部数据存储器。

当单片机的内部数据存储器不够用时，需要外扩数据存储器。MCS-51 最多可外扩 64KB 的 RAM 或 I/O。在实际应用中，扩展外部数据存储器容量的大小，由用户根据需要而定。MCS-51 单片机访问外部数据存储器是用一个特殊功能寄存器（16 位数据存储器地址指针）DPTR 进行的。由于 DPTR 为 16 位，所以可寻址的范围为 64KB。此外，还可以用 R0 和 R1 间接寻址片外低 256 个单元的数据存储器，通过修改寄存器 P2 的值，用 R0 和 R1 间接寻址片外数据存储器的范围也可以访问 64KB，即将 64KB 划分为 256 个区间，每个区间包含 256 个存储单元。

3. 并行 I/O 接口

MCS-51 单片机有 4 个 8 位并行 I/O 端口，分别记作 P0、P1、P2、P3，每个端口都各有 8 条 I/O 口线，共 32 条。P0~P3 口属于特殊功能寄存器范畴，这 4 个端口除了按字节寻址外，还可以按位寻址。

标准 51 内核单片机的 I/O 口：P0 口则为双向三态输入输出，P1、P2、P3 是准双向 I/O 口，没有方向控制，做输入时需要先往端口数据寄存器写 1 才行。

初始状态和复位状态下准双向口为 1，双向口为高阻状态。

这里特别要注意准双向与双向三态 I/O 的区别：

P1 口，P2 口，P3 口是 3 个 8 位准双向的 I/O 口，各口线在片内均有固定的上拉电阻器，当这三个准双向 I/O 口作输入口使用时，要向该口先写 1，另外准双向 I/O 口无高阻的“浮空”状态。而双向口 P0 口线内无固定上拉电阻器，由两个 MOS 管串接，既可开漏输出，又可处于高阻的“浮空”状态，故称为双向三态 I/O 口。

(1) P0 口

P0 口的位逻辑电路如图 1.5 所示。

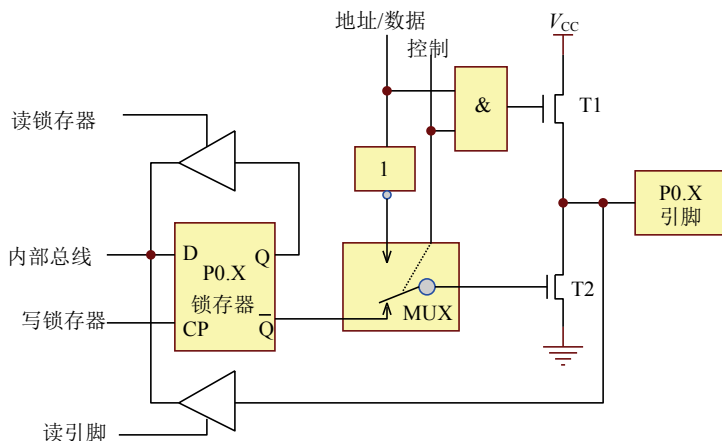


图 1.5 P0 口位逻辑电路原理图

P0 口是一个 8 位的三态双向数据总线口，可作为通用 I/O 接口使用，也可作为地址/数据线分时复用口使用，图 1.5 是 P0 口 1 位的结构原理图。它由一个输出锁存器，两个三态输入缓冲器，一个转换开关 MUX，一个输出驱动电路（T1 和 T2），一个与门及一个非

门组成。锁存器起输出锁存作用；两个三态输入缓冲器分别由“读引脚”和“读锁存器”两个不同的控制信号控制；场效应晶体管 T1、T2 组成输出驱动器；与门、非门及转换开关构成输出控制电路。

P0 口的工作方式由片内“控制”信号进行转换，当“控制”信号等于 0 时，MUX 转向 D 锁存器的反相输出端，P0 口作为通用 I/O 口；当“控制”信号等于 1 时，MUX 转向上方，P0 口作为地址/数据线分时复用口。转换开关的“控制”信号是由硬件根据操作指令自动产生的。

P0 口既可以作为地址/数据分时复用总线口，此时是一个真正的双向口，也可以作为通用的 I/O 接口，但只是一个准双向口。准双向口工作的特点是：当复位时，口锁存器均置 1，8 根引脚可当一般引脚线使用，而在某引脚由原输出状态变为输入时，则应先写入 1，以免错读引脚上的信息。此外，还可以通过“读—修改—写”类指令对端口实行“读—改—写”操作。

P0 口在实际应用中，一般情况下都是作为单片机系统的地址/数据分时复用总线口使用的，这时 P0 口就不能再作 I/O 口使用了。在作地址/数据分时复用总线口使用时，它分时输出低 8 位地址和传送数据信息，其输出的低 8 位地址要片外锁存，常用的办法是低 8 位地址与 ALE 信号配合予以实现。

(2) P1 口

P1 口只能作为通用 I/O 口使用，所以在电路结构上与 P0 口有一些不同之处，其位逻辑电路见图 1.6 所示。由于电路的内部有上拉电阻器，与场效应晶体管共同组成输出驱动电路。为此 P1 口作为输出口使用时，无需再外接上拉电阻器，这样电路的输出不是三态的。当 P1 口作为输入口使用时，同样也需先向其锁存器写 1，使输出驱动电路的 FET 截止。P1 口的输出级能驱动 4 个 LSTTL 负载。

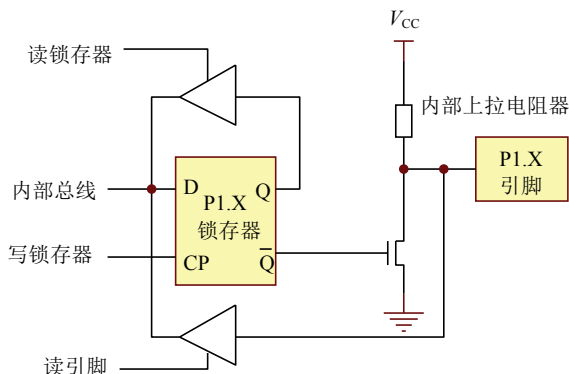


图 1.6 P1 口位逻辑电路原理图

(3) P2 口

P2 口电路中和 P0 口一样有一个多路转接电路 MUX，因此，P2 口既可以作为通用 I/O 口使用又可以作为地址端口使用。当多路转接开关倒向锁存器 Q 端时，P2 口可以作为通用 I/O 口使用，除内部有上拉电阻器不需外接上拉电阻器以外，其他与 P0 口作为通用口的使用方法相同。但通常应用情况下，P2 口作为高位地址线使用，此时多路转接开关应倒向地址线一端。P2 口的输出级能驱动 4 个 LSTTL 负载。P2 口的位逻辑电路见图 1.7。

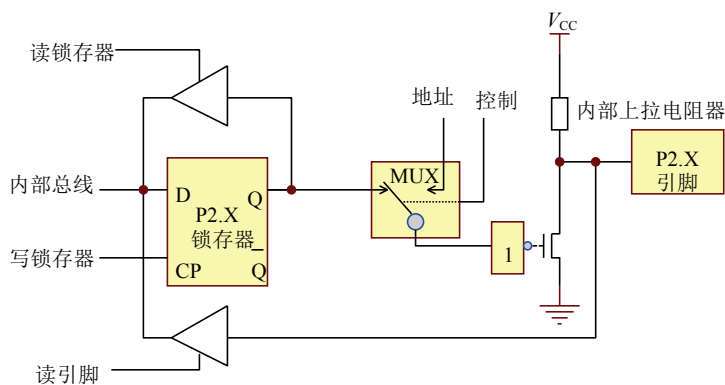


图 1.7 P2 口位逻辑电路原理图

(4) P3 口

P3 口的位逻辑电路见图 1.8。

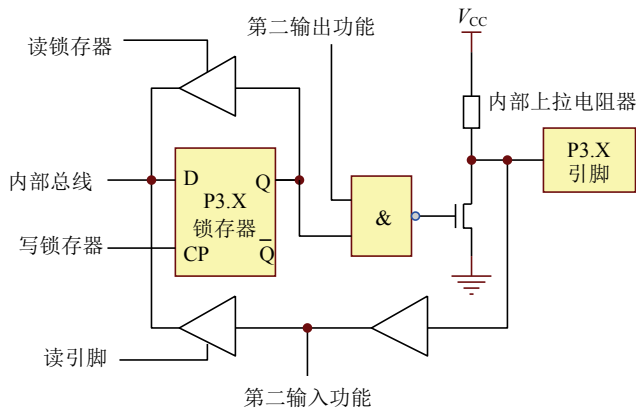


图 1.8 P3 口位逻辑电路原理图

由于 MCS-51 系列单片机的引脚有限，在系统扩展时，部分控制信号由 P3 口承担，因此，P3 口的特点在于增加了第二功能控制逻辑，如表 1.6 所示。由于第二功能信号有输入和输出两类，因此分两种情况说明。

表 1.6 P3 口引脚的第二功能

口 引 脚	第二功能	功能说明
P3.0	RXD	串行口输入
P3.1	TXD	串行口输出
P3.2	$\overline{INT0}$	外部中断 0 输入
P3.3	$\overline{INT1}$	外部中断 1 输入
P3.4	T0	定时/计数器 0 外部计数脉冲输入
P3.5	T1	定时/计数器 1 外部计数脉冲输入
P3.6	\overline{WR}	片外数据存储器和外部扩展 I/O 口写选通（输出）
P3.7	\overline{RD}	片外数据存储器和外部扩展 I/O 口读选通（输出）

对于第二功能为输出的信号引脚,当作 I/O 使用时,第二功能信号引线应保持高电平,与非门开通,以保持从锁存器到输出端数据输出通路的畅通。当输出第二功能信号时,该位的锁存器应置 1,使与非门对第二功能信号的输出是畅通的,从而实现第二功能信号的输出。

对于第二功能为输入的信号引脚,在口线的输入通路上增加了一个缓冲器,输入的第二功能信号就从这个缓冲器的输出端取得。而作为 I/O 使用的数据输入,仍取自三态缓冲器的输出端。不管作为输入口使用还是第二功能信号输入,输出电路中的锁存器输出和第二功能输出信号线都应保持高电平。P3 口的输出级能驱动 4 个 LSTTL 负载。

4. MCS-51 时钟电路与复位电路

(1) 时钟电路

① 振荡器与时钟电路。

单片机的内部时钟信号是由振荡器产生的振荡脉冲二分频得到的。振荡器产生的脉冲电路如图 1.9 (a) 所示。其中 C_1 和 C_2 起频率微调作用,外接石英晶体时选 30pF 左右,外接陶瓷谐振器时选 40pF 左右。晶体的振荡频率决定时钟电路的振荡频率,其频率范围一般在 0~33MHz 之间。单片机还可以采用外部时钟信号方式,如图 1.9 (b) 所示。

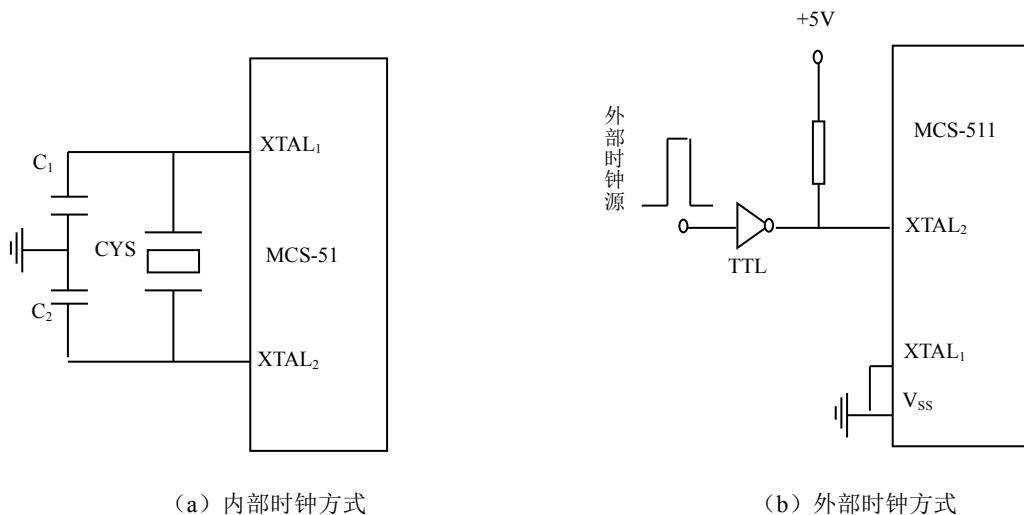


图 1.9 时钟电路

② 时序的基本概念。

单片机内的各种操作都是在一系列脉冲控制下进行的,而各脉冲在时间上是有先后顺序的,这种顺序就称为时序,它表明了指令执行中各种信号之间的相互关系。为了保证同步工作方式的实现,全部电路应在统一的时钟信号控制下严格地按时序进行工作。MCS-51 时序的基本定时单位共有四个,它们之间的关系如图 1.10 所示。

- 时钟振荡周期: 由振荡电路产生的振荡脉冲的周期,又称为节拍。
- S 状态周期: 也称为时钟周期,是时钟振荡周期的二倍宽。
- 机器周期: 通常将完成一个基本操作所需的时间称为机器周期,它是执行指令的

单位周期。

- 指令周期：执行一条指令（从取指令到指令执行完成）所需要的时间，一个指令周期通常含有 1~4 个机器周期。

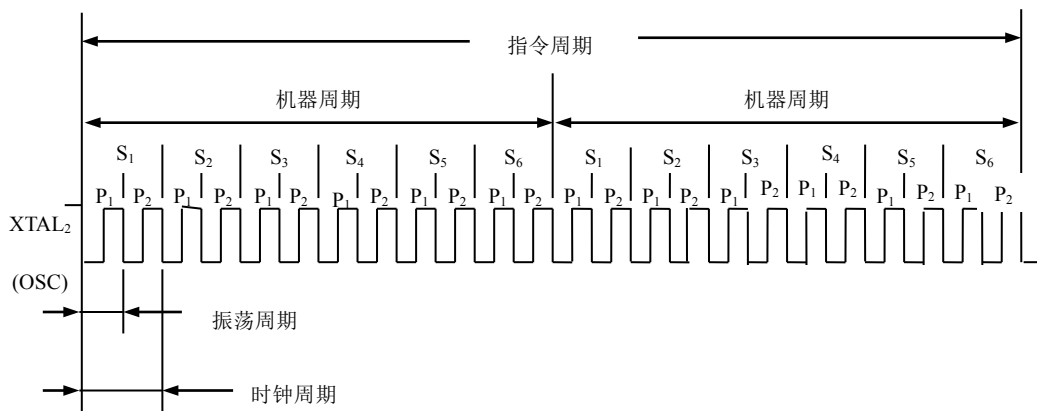


图 1.10 各种周期关系图

③ 时钟电路的振荡信号

8051 片内设有一个由反相放大器所构成的振荡电路， $XTAL_1$ 和 $XTAL_2$ 分别为振荡电路的输入和输出端，时钟可以由内部方式产生或外部方式产生。内部方式时钟电路如图 1.9 (a) 所示。在 $XTAL_1$ 和 $XTAL_2$ 引脚上外接定时元件，内部振荡电路就产生自激振荡。定时元件通常采用石英晶体和电容组成的并联谐振回路。晶振可以在 1.2MHz 到 12MHz 之间选择，电容量在 5~30pF 之间选择，电容器的大小可起频率微调作用。

外部方式的时钟很少用，若要用时，只要将 $XTAL_1$ 接地， $XTAL_2$ 接外部振荡器就行。对外部振荡信号无特殊要求，只要保证脉冲宽度，一般采用频率低于 12MHz 的方波信号。

时钟发生器把振荡频率二分频，产生一个两相时钟信号 P_1 和 P_2 供单片机使用。 P_1 在每个状态 S 的前半部分有效， P_2 在每个状态 S 的后半部分有效。

④ 脉冲分配器

MCS-51 典型的指令周期（执行一条指令的时间称为指令周期）为一个机器周期，一个机器周期由六个状态（十二振荡周期）组成。每个状态又被分成两个时相 P_1 和 P_2 。所以，一个机器周期可以依次表示为 S_1P_1 , S_1P_2 , ..., S_6P_1 , S_6P_2 。通常算术逻辑操作在 P_1 时进行，而内部寄存器传送在 P_2 时进行。

图 1.10 给出了 MCS-51 单片机的取指和执行指令的定时关系。这些内部时钟信号不能从外部观察到，所以用 $XTAL_2$ 振荡信号作参考。低 8 位地址的锁存信号 ALE 在每个机器周期中两次有效：一次在 S_1P_2 与 S_2P_1 期间，另一次在 S_4P_2 与 S_5P_1 期间。

(2) 单片机系统的复位

复位是单片机进入工作状态的初始化操作。当程序运行错误或由错误操作而使单片机进入死锁状态时，也可通过复位进行重新启动。

MCS-51 单片机的复位方式有上电自动复位和按键手动复位两种。按键手动复位又分为按键电平复位和按键脉冲复位两种。

① 上电自动复位

上电复位电路只需在 RST 端接一个电容器至 VCC 和一个电阻器至 VSS 即可,如图 1.11 所示。加电瞬间, RST 端出现一段时间的高电平, 只要高电平保持至少 2 个机器周期, 51 单片机就会执行复位操作, 然后把 RST 恢复为低电平。电路中, 时间常数 RC 越大, 上电时保持高电平的时间越长, 当振荡频率为 12 MHz 时, 典型值 $C=10\ \mu\text{F}$, $R=8.2\ \text{k}\Omega$, 当时钟频率选用 6 MHz 时, $C=22\ \mu\text{F}$, $R=1\ \text{k}\Omega$ 。

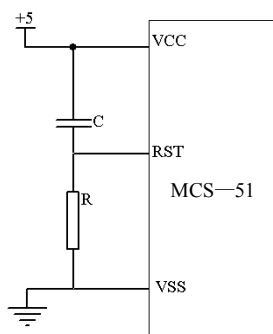


图 1.11 上电复位电路

若上电复位失效, 用户上电后 CPU 从一个随机状态开始工作, 系统则不能正常运行。上电复位后, RAM 单元数据是随机的。

② 人工复位

除了上电复位外, 有时还需要人工复位。将一个按钮开关并联于上电自动复位电路, 就是人工复位电路, 如图 1.12 所示。

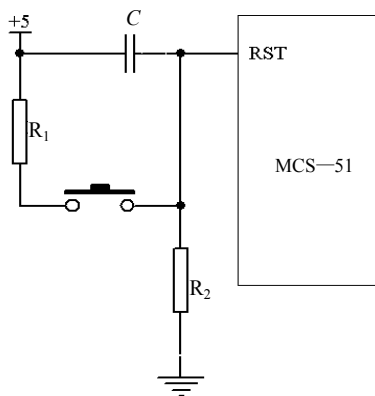


图 1.12 人工复位电路

按一下开关就会在 RST 端出现一段时间的高电平, 使单片机复位。电容量和电阻值的参考值是: $C=22\ \mu\text{F}$, $R_1=200\ \Omega$, $R_2=1\ \text{k}\Omega$ 。

③ 复位状态

单片机复位操作的主要作用是使 PC 值为 0000H, 这样单片机将从 0000H 单元开始执行程序。复位操作还会影响其他某些专用寄存器, 它们的状态见表 1.7。

表 1.7 专用寄存器复位状态

寄 存 器	状 态	寄 存 器	状 态
ACC	00H	TMOD	00H
B	00H	TCON	00H
PSW	00H	T2CON	00H
SP	07H	TH0	00H
DPTR:		TL0	00H
DPH	00H	TH1	00H
DPL	00H	TL1	00H
P0	FFH	TH2	00H
P1	FFH	TL2	00H
P2	FFH	RCAP2H	00H
P3	FFH	RCAP2L	00H
IP:		SCON	00H
8051	XXX00000B	SBUF	未定
8052	XX000000B	PCON:	
IE		NMOS	0XXXXXXXB
8051	0XX00000B	CHMOS	0XX00000B
8052	0X000000B	PC	00H

1.1.2 引脚功能

MCS-51 40 引脚双列直插式集成电路芯片，引脚排列见图 1.13。单片机的 40 个引脚大致可分为 4 类：电源、时钟、控制和 I/O 引脚。

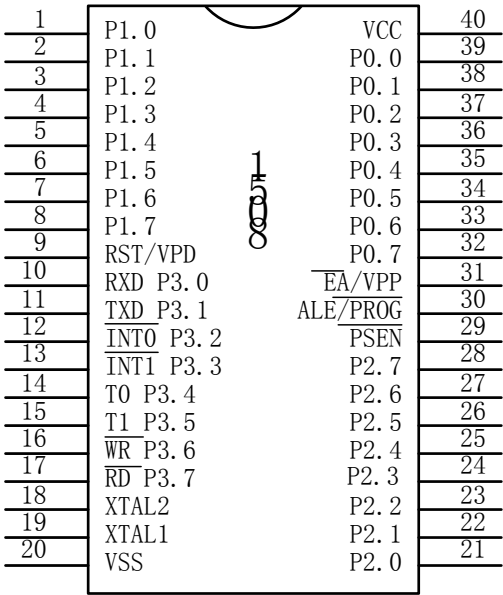


图 1.13 MCS-51（8051）芯片引脚排列

1. 电源

- (1) VCC (40) ——芯片电源，接+5V。
- (2) VSS (20) ——接地端。

2. 时钟

XTAL1 (19)、XTAL2 (18) ——晶体振荡电路反相输入端和输出端，当使用芯片内部时钟时，用于外接石英晶体和微调电容器；当使用外部时钟时，用于接外部时钟脉冲信号。当使用芯片内部时钟时，此二引线端用于外接石英晶体和微调电容器；当使用外部时钟时，用于接外部时钟脉冲信号。

3. 控制线（共 4 根）

- (1) ALE/ $\overline{\text{PROG}}$ (30) ——地址锁存允许/片内 EPROM 编程脉冲。

① ALE 功能：用来锁存 P0 口送出的低 8 位地址。在系统扩展时，ALE 用于控制把 P0 口输出的低 8 位地址锁存起来，以实现低位地址和数据的隔离。此外，由于 ALE 是以晶振 1/6 的固定频率输出的正脉冲，因此，可作为外部时钟或外部定时脉冲使用。

- ② $\overline{\text{PROG}}$ 功能：片内有 EPROM 的芯片，在 EPROM 编程期间，此引脚输入编程脉冲。

(2) $\overline{\text{PSEN}}$ (29) ——外部 ROM 读选通信号。在读外部 ROM 时， $\overline{\text{PSEN}}$ 有效（低电平），以实现外部 ROM 单元的读操作。

- (3) RST/VPD (9) ——复位/备用电源。

① RST (Reset) 功能：复位信号输入端。当输入的复位信号延续两个机器周期以上的高电平时即为有效，用以完成单片机的复位初始化操作。

- ② VPD 功能：在 VCC 掉电情况下，接备用电源。

- (4) $\overline{\text{EA}}$ /VPP (31) ——内外 ROM 选择/片内 EPROM 编程电源。

- ① $\overline{\text{EA}}$ 功能：内外 ROM 选择端。

- ② VPP 功能：片内有 EPROM 的芯片，在 EPROM 编程期间，施加编程电源 VPP。

4. I/O 引脚

- (1) P0.7~P0.0 (32~39)

P0 口是一个 8 位漏极开路的双向 I/O 口。作为输出口，每位能驱动 8 个 TTL 逻辑电平。

对 P0 端口写 1 时，引脚用作高阻抗输入。当访问外部程序和数据存储器时，P0 口也被作为低 8 位地址/数据复用。在这种模式下，P0 具有内部上拉电阻器。在 Flash 编程时，P0 口也用来接收指令字节；在程序校验时，输出指令字节，需要外部上拉电阻器。

P0 端口的第一功能是作为一个 8 位漏极开路型的双向 I/O 端口。第二功能是在访问外部存储器时，分时提供低 8 位地址和 8 位双向数据。在对单片机内部 EPROM 进行编程和校验时，P0 端口用于数据的输入和输出。

- (2) P1.7~P1.0 (8~1)

P1 口是一个具有内部上拉电阻器的 8 位双向 I/O 口，P1 输出缓冲器能驱动 4 个 TTL 逻辑电平。对 P1 端口写 1 时，内部上拉电阻器把端口拉高，此时可以作为输入口使用。作为输入使用时，被外部拉低的引脚由于内部电阻的原因，将输出电流（IIL）。此外，

P1.0 和 P1.2 分别作定时器/计数器 T2 的外部计数输入（P1.0/T2）和定时器/计数器 T2 的触发输入（P1.1/T2EX），具体功能见表 1.8。在 Flash 编程和校验时，P1 口接收低 8 位地址字节。

表 1.8 P1 接口第二功能

引脚号	第二功能
P1.0	T2（定时器/计数器 T2 的外部计数输入），时钟输出
P1.1	T2EX（定时器/计数器 T2 的捕捉/重载触发信号和方向控制）
P1.5	MOSI（在系统编程用）
P1.6	MISO（在系统编程用）
P1.7	SCK（在系统编程用）

（3） P2.7～P2.0（29～21）

P2 口是一个具有内部上拉电阻器的 8 位双向 I/O 口，P2 输出缓冲器能驱动 4 个 TTL 逻辑电平。对 P2 端口写“1”时，内部上拉电阻器把端口拉高，此时可以作为输入口使用。作为输入使用时，被外部拉低的引脚由于内部电阻的原因，将输出电流（IIL）。在访问外部程序存储器或用 16 位地址读取外部数据存储器（例如执行 MOVX @DPTR）时，P2 口送出高 8 位地址。在这种应用中，P2 口使用很强的内部上拉发送 1。在使用 8 位地址（如 MOVX @Ri）访问外部数据存储器时，P2 口输出 P2 锁存器的内容。在 flash 编程和校验时，P2 口也接收高 8 位地址字节和一些控制信号。

P2 端口的第一功能是作为一个内部带有上拉电阻器的 8 位准双向 I/O 端口。第二功能是在访问外部存储器时，输出高 8 位地址。

（4） P3.7～P3.0（17～10）

P3 口是一个具有内部上拉电阻器的 8 位双向 I/O 口，P2 输出缓冲器能驱动 4 个 TTL 逻辑电平。对 P3 端口写“1”时，内部上拉电阻器把端口拉高，此时可以作为输入口使用。作为输入使用时，被外部拉低的引脚由于内部电阻的原因，将输出电流（IIL）。P3 口亦可作为特殊功能（第二功能）使用。在 Flash 编程和校验时，P3 口也接收一些控制信号。

1.2 学习实例

实例一 LED 灯闪烁

1. 实例说明

要求让 AT89C51 的 P2 口接的 8 只发光二极管以一定的频率闪烁。编程时可先让 P2 口输出全“0”，延时一段时间后输出全“1”，循环。闪烁快慢由延时时间决定。

2. 仿真电路

LED 灯闪烁仿真电路如图 1.14 所示。

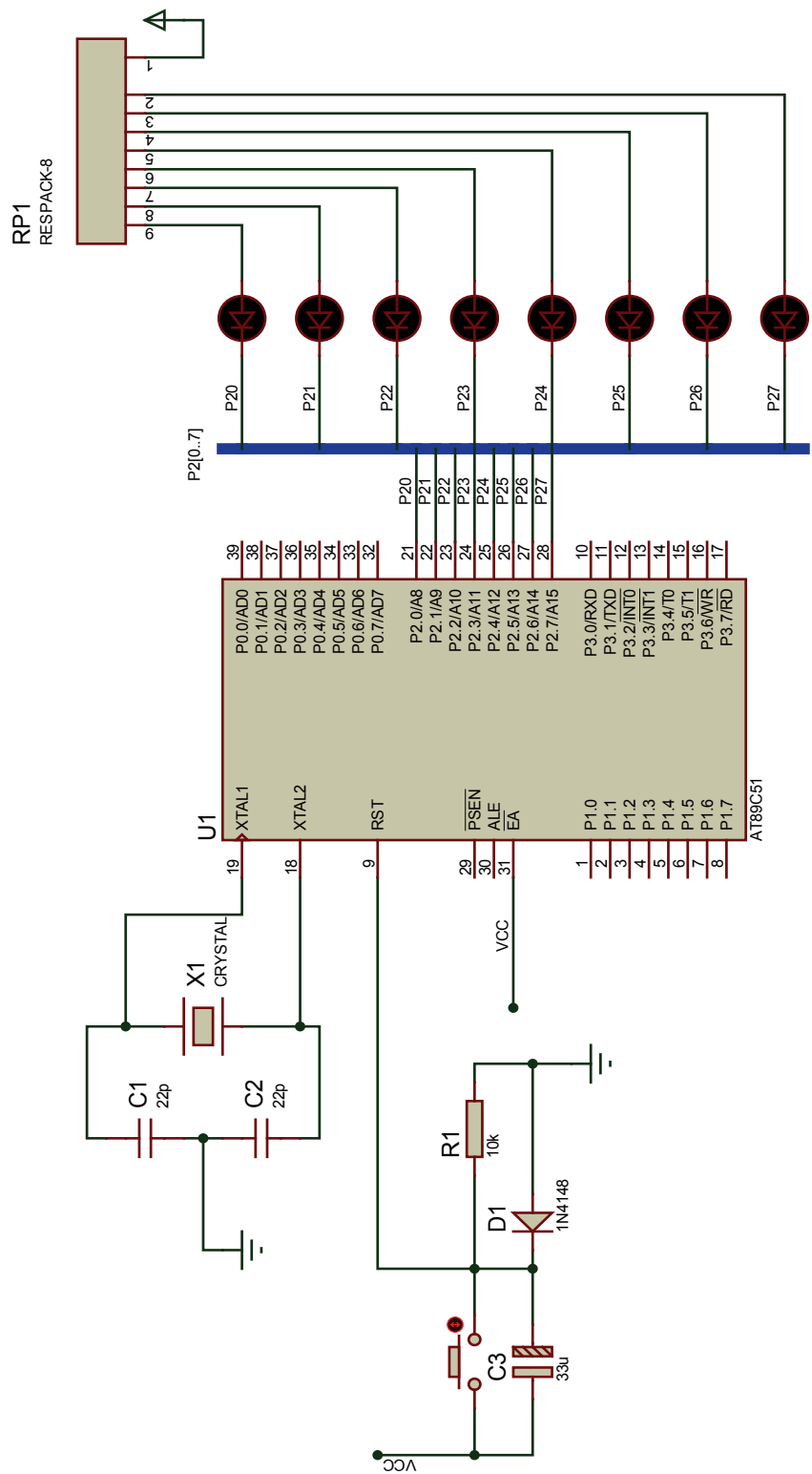


图 1.14 LED 灯闪烁仿真电路

3. 程序设计

```
#include <reg51.h>      //此文件中定义了 51 的一些特殊功能寄存器
void delay(unsigned int i); //延时函数声明
main()
{
    while(1)
    {
        P2 = 0x00; //置 P0 口为低电平
        delay(600); //调用延时程序
        P2 = 0xff; //置 P0 口为高电平
        delay(600); //调用延时程序
    }
}
/*****延时函数*****/
void delay(unsigned int i)
{
    unsigned char j;
    for(i; i > 0; i--) //循环 600*255 次
        for(j = 255; j > 0; j--);
}
```

4. 用开发板实验

本例程序编译通过后，生成 HEX 文件，烧录到 STC89C51 单片机中，插入实验板，通电运行观察显示效果。

实例二 LED 流水灯

1. 实例说明

要求让 P1 口接的八只发光二极管循环依次点亮，每次亮一个。7406 是反相驱动器，请查阅有关手册。仿真时省去了时钟和复位电路。

2. 仿真电路

LED 流水灯仿真电路如图 1.15 所示。

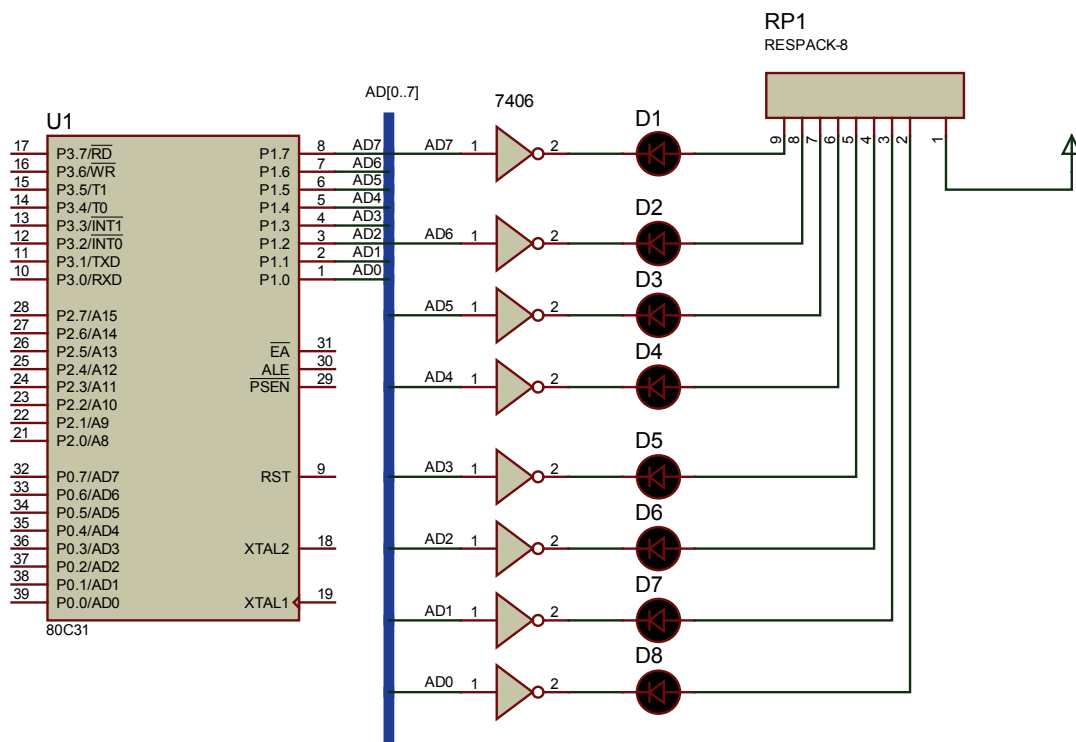


图 1.15 LED 流水灯仿真电路

3. 程序设计

```
#include<reg51.h>
#include<intrins.h>
void delay(void) //延时
{
    unsigned int i,j;
    for(i=0;i<500;i++)
        for(j=0;j<250;j++)
            ;
}
main()
{
    unsigned char LED;
    LED=0x01; //初值
    P1=LED; //送 P1 口
    while(1)
    {
        delay();
        LED=_crol_(LED,1); //左移
        P1=LED;
    }
}
```

实例三 转向灯

1. 实例说明

要求实现转向灯的效果，LEFT 和 RIGHT 开关同时打在上方和同时打在下方时两个灯都不亮，当 LEFT 开关打在上方，RIGHT 打在下方时，LEFT-LAMP 以一定的频率闪烁，同样，当 LEFT 开关打在下方，RIGHT 打在上方时，RIGHT-LAMP 以一定的频率闪烁。

2. 仿真电路

转向灯仿真电路如图 1.16 所示。

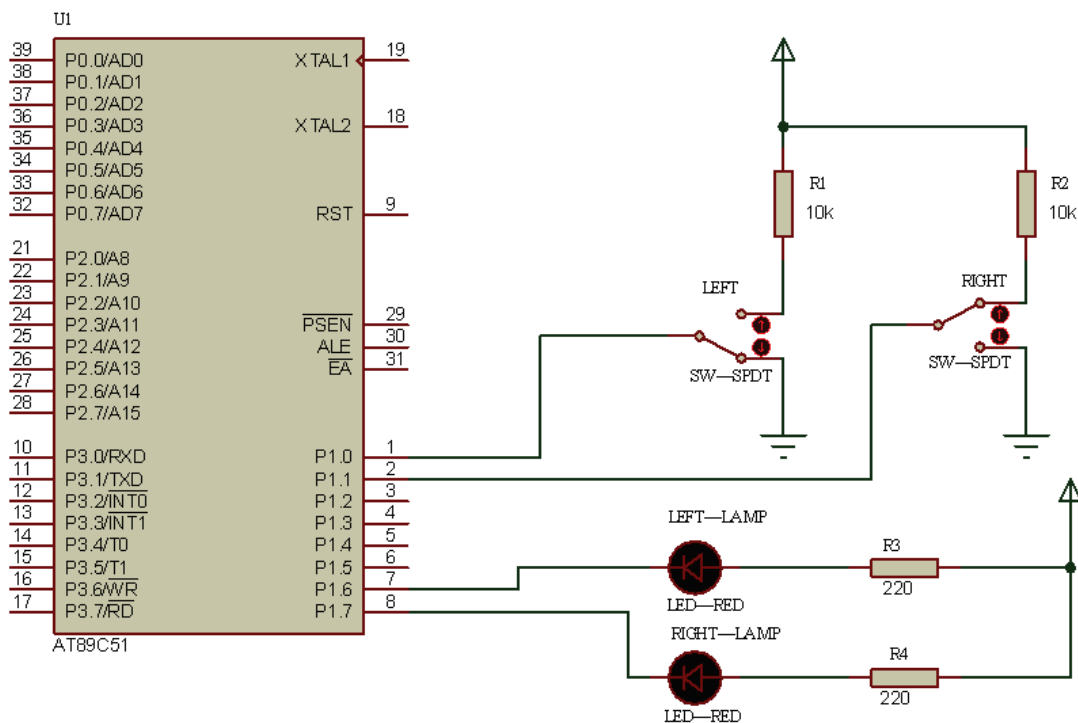


图 1.16 转向灯仿真电路

3. 程序设计

```
#include <reg51.h>
sbit L=P1^0 //位定义
sbit R=P1^1;
sbit LL=P1^6;
sbit RL=P1^7;
void delay60ms( ) //延时
{
    unsigned int m,n;
    for(m=0;m<300;m++)
```

```

        for(n=0;n<300;n++) ;
    }
    void leftlamp(void)  //左转向灯闪烁
    {
        LL=0;
        delay60ms();
        LL=1;
        delay60ms();
    }
    void rightlamp(void) //右转向灯闪烁
    {
        RL=~RL;
        delay60ms();
        RL=1;
        delay60ms();
    }
    main(void)
    {
        while(1)
        {
            if(L==1&&R==0)
                leftlamp(); //函数调用
            if(L==0&&R==1)
                rightlamp();
        }
    }
}

```

本章小结

- (1) 主要介绍了 51 单片机的内部结构和各引脚名称及功能。
- (2) 就驱动能力来说，P0 口的驱动能力最强，为 8 个 TTL 门，其余 3 个口驱动能力为 4 个 TTL 门，在接口使用时应注意其负载能力。
- (3) 了解单片机的存储器结构，为后面的编程和应用打基础。
- (4) 复位对单片机来说是一个很重要的概念，涉及到单片机的初始化，可靠性，故障处理等诸多方面。

习题一

一、填空题

1. MCS-51 单片机的 P0~P3 口均是_____ I/O 口，其中的 P0 口和 P2 口除了可以进行数据的输入、输出外，通常还用来构建系统的_____和_____，在 P0~P4 口中，_____为真正的双向口，_____为准双向口。
2. MCS-51 有 4 组工作寄存器，它们的地址范围是_____。
3. MCS-51 片内 20H~2FH 范围内的数据存储器，既可以字节寻址又可以_____。

寻址。

4. PSW 中 RS1 RS0=10H, R2 的地址为_____。
5. MCS-51 单片机访问片外部存储器时, 利用_____信号锁存来自 P0 口的低 8 位地址信号。
6. 在 MCS-51 中 PC 和 DPTR 都用于提供地址, PC 是为访问_____存储器提供地址, 而 DPTR 是为访问_____存储器提供地址。
7. MCS-51 复位后, PC 与 SP 的值分别为_____和_____。

二、单项选择题

1. 单片机 8051 的 XTAL1 和 XTAL2 引脚是 () 引脚。
A. 外接定时器
B. 外接串行口
C. 外接中断
D. 外接晶振
2. 片内 RAM 的 20H~2FH 为位寻址区, 所包含的位地址是 ()。
A. 00H~20H
B. 00H~7FH
C. 20H~2FH
D. 00H~FFH
3. 51 单片机的堆栈区通常设置在 () 中。
A. 片内 ROM 区
B. 片外 ROM 区
C. 片内 RAM 区
D. 片外 RAM 区
4. 关于 MCS-51 的堆栈操作, 正确的说法是 ()。
A. 先入栈, 再修改栈指针
B. 先修改栈指针, 再出栈
C. 先修改栈指针, 再入栈
D. 以上都不对
5. 当 ALE 信号有效时, 表示 ()。
A. 从 ROM 中读取数据
B. 从 P0 口可靠地送出地址低 8 位
C. 从 P0 口送出数据
D. 从 RAM 中读取数据
6. 8051 单片机中, 唯一一个用户不能直接使用的寄存器是 ()。
A. PSW
B. DPTR
C. PC
D. B
7. 8031 单片机的 () 口的引脚, 还具有外中断、串行通信等第二功能。
A. P0
B. P1
C. P2
D. P3
8. 8051 与 8751 的区别是 ()。
A. 内部数据存储器的数目不同
B. 内部程序存储器的类型不同
C. 内部数据存储器的类型不同
D. 内部寄存器的数目不同
9. PC 的值是 ()。
A. 当前正在执行指令的前一条指令的地址
B. 当前正在执行指令的下一条指令的地址
C. 当前正在执行指令的地址
D. 控制器中指令寄存器的地址
10. 单片机的堆栈指针 SP 始终是 ()。
A. 指示堆栈栈底
B. 指示堆栈栈顶
C. 指示堆栈地址
D. 指示堆栈长度

三、简答题

1. MCS-51 外扩的程序存储器和数据存储器可以有相同的地址空间，但不会发生数据冲突，为什么？
2. 说明 MCS-51 的外部引脚 \overline{EA} 的作用。
3. 8051 的 \overline{PSEN} 、 \overline{RD} 、 \overline{WR} 引脚的作用是什么？
4. ALE 引脚的作用是什么？当 8051 不和 RAM/ROM 相连时，ALE 的输出频率是多少？
5. MCS-51 的工作寄存区包含几个通用工作寄存器组？每组的地址是什么？如何选用？开机复位后，CPU 使用的是哪组工作寄存器？
6. MCS-51 的内部 RAM 地址空间是如何安排的？共有多少个单元可以位寻址？位地址又是如何排列的？
7. MCS-51 的程序计数器 PC 是几位寄存器？它是否为专用寄存器？PC 的内容是什么信息？
8. 什么是堆栈？堆栈指针 SP 的作用是什么？在程序设计时，有时为什么要对 SP 重新赋值？如果 CPU 在操作中要使用两组工作寄存器，SP 的初值应设为多少？
9. MCS-51 单片机的时钟周期，机器周期，指令周期是如何设置的？当振荡频率为 6MHz 时，计算一个机器周期和执行一条最长的指令各需多少时间？
10. 使单片机复位有几种方法？复位的条件是什么？复位后片内各寄存器及 RAM 的状态如何？



第2章 单片机汇编语言与C语言程序设计基础

学习目标

掌握 MCS-51 单片机指令格式、分类及寻址方式；掌握汇编语言程序设计流程及 C51 程序设计基本方法。

重点难点

- (1) 寻址方式、控制转移、位操作等指令的灵活应用。
- (2) 汇编语言、C 语言程序设计的基本方法和针对具体的硬件设计出对应程序的方法。

2.1 知识结构

MCS-51 单片机的编程语言可以是汇编语言，也可以是高级语言（如 C 语言），高级语言编程快捷，但程序长，占用存储空间大，执行慢；汇编语言产生的目标程序简短，占用存储空间小，执行快，能充分发挥计算机的硬件功能。无论是高级语言还是汇编语言，源程序都要转换成目标程序（机器语言），单片机才能执行。

2.1.1 汇编语言程序设计

2.1.1.1 寻址方式

操作数是指令的一个重要组成部分，CPU 在规定的寻址空间获得操作数地址的方式，称为寻址方式。寻址方式不仅影响指令的长度，还影响指令的执行速度。MCS-51 系列单片机的指令系统有立即寻址、直接寻址、寄存器寻址、间接寻址、变址寻址、相对寻址和位寻址等 7 种寻址方式。

1. 立即寻址

操作数作为指令的一个组成部分存放在程序存储器中。这种寻址方式称为立即寻址，该操作数称为立即数。立即数前加“#”标记，以便和直接寻址方式相区分。

例如指令：MOV A, #5AH; A←5AH

2. 直接寻址

直接寻址方式是在指令代码中直接给出操作数的地址，这种寻址方式是对内部数据存储器进行访问。

直接寻址方式可寻址的空间如下。

① 内部 RAM 的低 128 字节单元地址空间；特殊功能寄存器 SFR 地址空间（直接寻址是访问 SFR 的唯一方式）；

② 位地址空间。

例如指令：MOV A, 4AH; $A \leftarrow (4AH)$

该指令是把内部 RAM 中 4AH 单元（直接寻址）的内容送入累加器 A 中。假设指令执行前 A 的内容为 7BH[表示为 $(A)=7BH$]，4AH 单元的内容为 26H[表示为 $(4AH)=26H$]，则指令执行后： $(A)=26H$ ， $(4AH)=26H$ （不变），如图 2.1 所示。

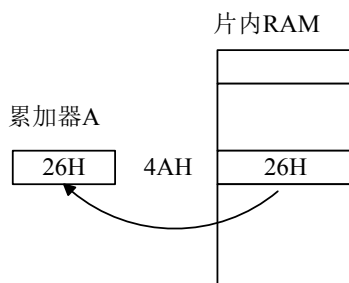


图 2.1 直接寻址示意图

3. 寄存器寻址

寄存器寻址方式是指指令中给出寄存器名称，其内容作为操作数。这种寻址方式对当前工作寄存器 R0~R7 进行访问，指令操作码的低三位指明了所用的寄存器；对累加器 A，寄存器 B，数据指针 DPTR 也可以用寄存器寻址来访问。

寄存器寻址方式的可寻址空间：

① 当前工作寄存器 R0~R7（当前工作寄存器区由 RS0、RS1 来选定）；

② 累加器 A、寄存器 B、数据寄存器 DPTR。

例如指令：MOV A, R2; $A \leftarrow (R2)$

该指令是将工作寄存器 R2 的内容送给累加器 A。假设指令执行前 $(A)=08H$ ， $(R2)=4EH$ ，则指令执行后： $(A)=4EH$ ， $(R2)=4EH$ 。

4. 间接寻址

指令中给出寄存器，以寄存器的内容作为操作数的地址，把该地址对应单元的内容作为操作数。

寄存器间接寻址的可寻址空间：

① 内部 RAM 00H~7FH ($@R0$ 、 $@R1$ 、SP)；

② 外部 RAM 0000H~FFFFH ($@R0$ 、 $@R1$ 、 $@DPTR$)。

在访问外部 RAM 的页内 256 个单元 $\times\times 00H \sim \times\times FFH$ 时，用 R0, R1 工作寄存器间接寻址。

在访问外部 RAM 整个 64KB (0000H~FFFFH) 地址空间时，用数据指针 DPTR 来间接寻址。

例如指令：MOV A, R0; $A \leftarrow (R0)$

```
MOVX A, @R0; A ← ((R0))
```

```
MOVX A, @DPTR; A ← ((DPTR))
```

第一条指令 R0 是寄存器寻址, 设 (R0) = 3AH, 指令执行后累加器 A 中为 3AH; 第二条指令是寄存器间接寻址, 该指令将寄存器 R0 的内容 3AH 作为地址, 把片内 RAM 3AH 单元的内容 [设 (3AH) = 65H] 送入累加器 A, 指令执行后 (A) = 65H。指令中 “@” 表示寄存器间接寻址, 称之为间址符。寄存器间接寻址示意图如图 2.2 所示。

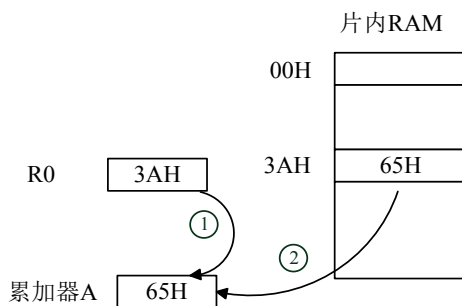


图 2.2 寄存器间接寻址示意图

5. 变址寻址

变址寻址是 MCS-51 系列单片机指令系统所特有的一种寻址方式。它以程序计数器 PC 或数据指针 DPTR 作为基址寄存器, 以累加器 A 作为变址寄存器 (存放 8 位无符号的偏移量), 两者内容相加形成 16 位程序存储器地址作为指令操作数的地址。这种寻址方式用于读取程序存储器中的常数表。

变址寻址空间: 程序存储器 (@A+DPTR, @A+PC)。

例如指令: `MOVC A, @A+DPTR; A ← ((A) + (DPTR))`

该指令是把 DPTR 的内容作为基地址, 把 A 的内容作为偏移量, 两者相加形成 16 位地址, 将程序存储器 ROM 中该地址单元的内容送给 A。假设指令执行前为: (DPTR) = 2100H, (A) = 56H, (2156H) = 36H, 则该指令执行后: (A) = 36H。

6. 相对寻址

相对寻址方式只用于相对转移指令中。相对转移指令是以本指令的下一条指令的首地址 PC 为基地址, 与指令中给定的相对偏移量 rel 相加之和作为程序的转移目标地址。偏移量 rel 是 8 位二进制补码 (与 PC 相加时, rel 需符号扩展成 16 位)。转移范围为当前 PC 值的 -128 ~ +127 个字节单元之间。相对寻址一般为双字节或三字节指令。

相对寻址空间: 程序存储器。

例如指令: `JZ 30H`; 当 (A) = 0 时, 则 $PC \leftarrow (PC) + 2 + rel$, 程序转移。

当 (A) = 1 时, 则 $PC \leftarrow (PC) + 2$ 。程序按原顺序执行。

该指令为双字节指令, 其执行过程示意图如图 2.3 所示。

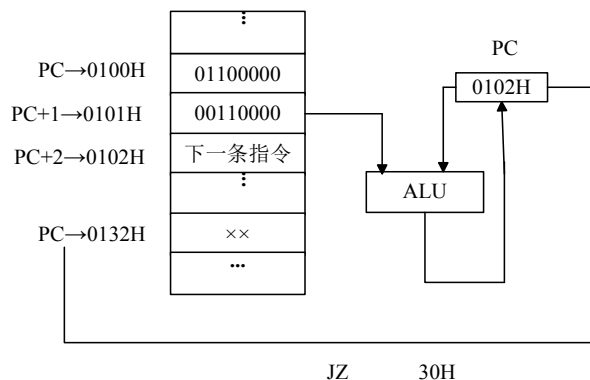


图 2.3 相对寻址执行过程示意图

7. 位寻址

位寻址是指按位进行的寻址操作，而上述介绍的指令都是按字节进行的寻址操作。MCS-51 单片机中，操作数不仅可以按字节为单位进行操作，也可以按位进行操作。当把某一位作为操作数时，这个操作数的地址就是位地址。表示位数据的方式有：直接使用该位的地址、使用字节地址加点及位数、使用寄存器名加点及位数、使用该位的名称。

位寻址空间：

(1) 片内 RAM 的位寻址区域是：20H~2FH 共 16 个单元 128 位，其位地址为 00H~7FH；

(2) 字节地址能被 8 整除的 SFR (12 个)。对这些寻址位，可以有以下四种表示方法：

① 直接位地址方式，如：0D5H；

② 位名称方式，如：F0；

③ 点操作符方式，如 PSW·5 或 0D0H·5；

④ 用户定义名方式，如用伪指令 bit 定义 USR_FLG bit F0，经定义后，允许指令用 USR_FLG 代替 F0。

以上四种方式指的都是 PSW 中的第 5 位。

MCS-51 单片机中设有独立的位处理器。位操作命令能对可以位寻址的空间进行位操作。

例如指令：MOV C, 07H ; Cy ← (07H)。

该指令属位操作指令，将内部 RAM 的 20H 单元的 07 位（位地址为 07H）的内容送给位累加器 Cy。

2.1.1.2 指令系统

1. 指令系统说明

(1) 分类

MCS-51 单片机的指令系统共有 111 条指令，按功能可以分为五大类。

① 数据传送指令（29 条）；

② 算术运算指令（24 条）；

③ 逻辑运算指令 (24 条);

④ 控制转移指令 (22 条);

⑤ 位操作指令 (12 条)。

(2) 指令说明

对于每一条指令, 学习时要注意掌握以下几点:

① 指令的功能;

② 指令操作数的合法寻址方式;

③ 指令对标志的影响;

④ 指令的长度和执行时间。

(3) 指令中符号的约定

在介绍汇编指令系统时, 指令中的符号约定如下:

Rn ($n=0\sim7$): 当前选中的工作寄存器组的工作寄存器 $R0\sim R7$ 。

@Ri ($i=0, 1$): 以 $R0$ 或 $R1$ 作寄存器间接寻址, “@”为间址符。

direct: 8 位直接地址。可以是内部 RAM 单元地址 ($00H\sim7FH$), 或特殊功能寄存器 (SFR) 地址 ($80H\sim FFH$)。

@DPTR: 以数据指针 DPTR 的内容 (16 位) 为地址的寄存器间接寻址, 对外部 RAM 的 64KB 地址空间寻址。

#data: 8 位立即数 “#” 表示 DATA 是立即数, 不是直接寻址。

#data16: 16 位立即数。

addr11: 11 位直接地址。绝对转移 (AJMP) 及绝对调用 (ACALL) 指令中的转移目标地址, 2KB 范围内转移, 指令中常用标号代替。

addr16: 16 位直接地址。长转移 (LJMP) 及长调用 (LCALL) 指令中的转移目标地址, 转移范围 64KB, 指令中常用标号代替。

rel: 相对转移地址, 8 位补码, 地址偏移量为 $-128\sim+127$, 指令中常用标号代替。

DPTR: 16 位数据指针。

bit: 位地址。内部 $RAM20H\sim2FH$ 中可寻址位和 SFR 中的可寻址位。

ACC: 累加器。

B: B 寄存器。

C: 即 Cy 位, 进位/借位标志位, 或位操作指令中的位累加器。

/: 位操作数的取反操作前缀, 如/bit。

(X): X 中的内容。

((X)): 由 X 间接寻址的单元中的内容。

←: 箭头右边内容是箭头所指的单元。

2. 数据传送类指令

数据传送指令共有 29 条, 分为片内寄存器, 数据存储器传送 (MOV) 指令, 片外数据存储器传送 (MOVB) 指令, 程序存储器内查表 (MOVC) 指令, 数据交换 (XCH、XCHD、SWAP) 指令和堆栈操作 (PUSH、POP) 指令。源操作数可以采用寄存器, 寄存器间接, 直接, 立即, 变址 5 种方式, 目的操作数可以采用前 3 种寻址方式。数据传送指令是编程

时使用最频繁的一类指令。

数据传送指令不影响标志。这里所说的标志是指 C, AC 和 OV, 不包括检验累加器奇偶的标志 P。对于 P 一般不加说明。

(1) 片内数据 RAM 及寄存器间的数据传送指令 (16 条)

① 以累加器 A 为目的的操作数的指令 (4 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV A, Rn	将 Rn 中内容 (简称 Rn 值, 下同) 送入 A 中, $A \leftarrow (Rn)$	1	12	1
MOV A, direct	将 direct 单元中内容 (简称 direct 值, 下同) 送入 A 中, $A \leftarrow (direct)$	2	12	1
MOV A, #data	将 8 位常数 #data 送入 A 中, $A \leftarrow data$	2	12	1
MOV A, @Ri	将 Ri 间址的地址单元中内容 (简称 Ri 间址值, 下同) 送入 A 中, $A \leftarrow (Ri)$	1	12	1

② 以 Rn 为目的的操作数的指令 (3 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV Rn, direct	将 direct 值送入 Rn 中, $Rn \leftarrow (direct)$	2	24	2
MOV Rn, #data	将常数 data 送入 Rn 中, $Rn \leftarrow data$	2	12	1
MOV Rn, A	将 A 值送入 Rn 中, $Rn \leftarrow (A)$	1	12	1

③ 以直接地址为目的的操作数的指令 (5 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV direct, Rn	将 Rn 值送入 direct 中, $direct \leftarrow (Rn)$	2	24	2
MOV direct, A	将 A 值送入 direct 中, $direct \leftarrow (A)$	2	12	1
MOV direct, @Ri	将 Ri 间址值送入 direct 中, $direct \leftarrow (Ri)$	2	24	2
MOV direct, #data	将常数 #data 直接送入 direct 中, $direct \leftarrow data$	3	24	2
MOV direct1, direct2	将 direct1 值送入 direct2 中, $direct1 \leftarrow (direct2)$	3	24	2

④ 以间接地址为目的的操作数的指令 (3 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV @Ri, A	将 A 值送入 Ri 指示的地址单元中, $(Ri) \leftarrow (A)$	1	12	1
MOV @Ri, direct	将 direct 值送入 Ri 指示的地址单元中, $(Ri) \leftarrow (direct)$	2	24	2
MOV @Ri, #data	将常数 #data 直接送入 Ri 指示的地址单元中, $(Ri) \leftarrow data$	2	12	1

⑤ 16 位数据传送指令 (1 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV DPTR, #DATA16	将常数 DATA16 直接送入 DPTR 中, $DPTR \leftarrow DATA16$	3	24	2

该条指令是整个指令系统中唯一的一条 16 位数据传送指令,通常用来设置地址指针, DPTR 由 DPH 和 DPL 组成,该指令传送时,把高 8 位立即数送入 DPH,低 8 位立即数送入 DPL 中。

MOV 指令用于寻址内部 RAM 和 SFR, MOV 指令对内部 RAM 和 SFR 的操作功能可用图 2.4 所示来描述。

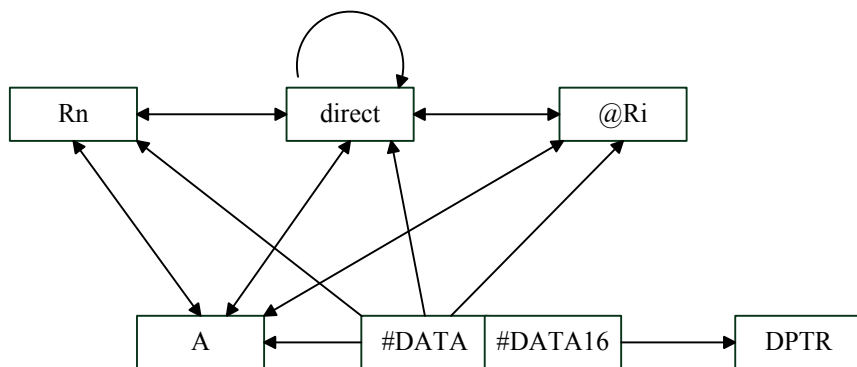


图 2.4 MOV 数据传送指令

(2) 堆栈操作指令 (2 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
PUSH direct	入栈指令, 将 direct 值压入堆栈栈顶, $sp \leftarrow (sp) + 1$, $(sp) \leftarrow (direct)$	2	24	2
POP direct	出栈指令, 将堆栈栈顶内容弹出到 direct 中, $direct \leftarrow (sp)$, $sp \leftarrow (sp) - 1$	2	24	2

(3) 数据交换指令 (5 条)

数据交换指令有字节交换指令和半字节交换指令两种。字节交换指令可以将累加器 A 的内容与内部 RAM 中任一个单元以字节为单位进行交换; 半字节交换指令可以将累加器 A 的高半字节和低半字节进行交换, 或将累加器 A 的低半字节与 @Ri 寻址单元的低半字节相互交换。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
XCH A, Rn	Rn 值和 A 值全交换, $(A) \longleftrightarrow (Rn)$	1	12	1
XCH A, direct	direct 值与 A 值全交换, $(A) \longleftrightarrow (direct)$	2	12	1
XCH A, @Ri	@Ri 值与 A 值全交换, $(A) \longleftrightarrow (Ri)$	1	12	1
XCHD A, @Ri	@Ri 与 A 值的低四位交换, $(A)_{3-0} \longleftrightarrow (Ri)_{3-0}$	1	12	1
SWAP A	A 值自交换 (高 4 位与低 4 位交换), $(A)_{7-4} \longleftrightarrow (A)_{3-0}$	1	12	1

(4) 片外 RAM 数据传送指令 (4 条)

片外 RAM 的数据传送指令助记符为 MOVX。片外数据存储器为读写存储器, 与累加器 A 可实现双向操作。片外数据存储器 (RAM) 使用寄存器间接寻址。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOVX A, @DPTR	用 DPTR 间址的片外 RAM 指定单元内容送入 A 中, $A \leftarrow (DPTR)$	1	24	2
MOVX @DPTR, A	将 A 中的内容送至片外 RAM 的 DPTR 间址的单元中, $(DPTR) \leftarrow (A)$	1	24	2
MOVX A, @Ri	将片外 RAM 中由 Ri 间址的地址单元中内容送入 A 中, $A \leftarrow (Ri)$	1	24	2
MOVX @Ri, A	将 A 中的内容送至片外 RAM 中由 Ri 间址的地址单元中, $(Ri) \leftarrow A$	1	24	2

注意

MSC-51 扩展的 I/O 端口地址占用的是片外 RAM 的地址空间, 因此对扩展的 I/O 接口而言, 这 4 条指令为输入/输出 (I/O) 指令。MCS-51 只能用这种方式与扩展的 I/O 口进行数据传送。

指令中以 DPTR 中的内容为片外 RAM 的 16 位地址指针时, 由 P0 口送出低 8 位地址, 由 P2 口送出高 8 位地址, 寻址范围为 64KB; 以 R0 或 R1 为片外 RAM 的低 8 位地址指针时, 由 P0 口送出 8 位地址, P2 口的状态不变化, 寻址范围为 256 个单元。

(5) 程序存储器查表指令 (2 条)

程序存储器查表指令的助记符为 MOVC。程序存储器为只读存储器。程序运行中所需的一些常数和表格, 通常由用户先将其写在程序存储器中, 程序需从程序存储器中读出数据时, 采用变址寻址方式将表格常数读入累加器 A 中。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOVC A, @A+DPTR	将以 DPTR 为基址, A 为偏移地址的存储单元内容送入 A 中, $PC \leftarrow (PC) + 1$, $A \leftarrow ((A) + (PC))$	1	24	2
MOVC A, @A+PC	将以 PC 为基址, A 为偏移地址的存储单元内容送入 A 中, $A \leftarrow ((A) + (DPTR))$	1	24	2

3. 算术运算指令

MSC-51 的算术运算指令共有 24 条。分为:

加法指令 (ADD, ADDC, INC);

带借位减法指令 (SUBB, DEC);

乘除指令 (MUL, DIV) 和十进制数调整指令 (DA) 等。

MSC-51 算术运算指令能直接执行 8 位数的运算, 借助程序状态字 PSW 中的标志可以实现多精度数的加、减运算, 同时可以对压缩的 BCD (一个字节表示两位十进制数) 数进行加法运算并调整。

算术运算指令对程序状态字 (PSW) 中标志位的影响如下面所示。

指 令	PSW 中的标志位		
	Cy	OV	AC
ADD	×	×	×
ADDC	×	×	×
INC	—	—	—
SUBB	×	×	×
DEC	—	—	—
MUL	0	×	—
DIV	0	×	—

“×”表示影响该标志位，“—”表示不影响该标志位，“0”表示该标志位清零。

(1) 加法指令

① 不带进位的加法指令（4条）。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
ADD A, Rn	Rn 值与 A 值相加, 结果在 A 中, $A \leftarrow (A) + (Rn)$	1	12	1
ADD A, direct	Direct 值与 A 值相加, 结果在 A 中, $A \leftarrow (A) + (direct)$	2	12	1
ADD A, #data	常数 data 与 A 值相加, 结果在 A 中, $A \leftarrow (A) + data$	2	12	1
ADD A, @Ri	@Ri 值与 A 值相加, 结果在 A 中, $A \leftarrow (A) + ((Ri))$	1	12	1

② 带进位加法指令（4条）。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
ADDC A, Rn	Rn 值与 A 值带进位加, 结果送 A, $A \leftarrow (A) + (Rn) + (Cy)$	1	12	1
ADDC A, direct	direct 值与 A 值带进位加, 结果送 A, $A \leftarrow (A) + (direct) + (Cy)$	2	12	1
ADDC A, #data	常数 data 与 A 值带进位加, 结果送 A, $A \leftarrow (A) + DATA + (Cy)$	2	12	1
ADDC A, @Ri	Ri 间址的存储单元中内容与 A 值带进位加, 结果送 A, $A \leftarrow (A) + (Ri) + (Cy)$	1	12	1

③ 加 1 指令（5条）。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
INC A	A 值加 1, $A \leftarrow (A) + 1$	1	12	1
INC Rn	Rn 值加 1, $Rn \leftarrow (Rn) + 1$	1	12	1
INC direct	direct 值加 1, $direct \leftarrow (direct) + 1$	2	12	1
INC @Ri	@Ri 值加 1, $((Ri)) \leftarrow ((Ri)) + 1$	1	12	1
INC DPTR	DPTR 值加 1, $DPTR \leftarrow (DPTR) + 1$	1	24	2

(2) 减法指令

① 带借位的减法指令（4条）。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
SUBB A, Rn	A 值减去 Rn 及 Cy 值, 结果送 A, $A \leftarrow (A) - (Rn) - (Cy)$	1	12	1
SUBB A, direct	A 值减去 direct 及 Cy 值, 结果送 A, $A \leftarrow (A) - (direct) - (Cy)$	2	12	1
SUBB A, #data	A 值减去 data 及 Cy 值, 结果送 A, $A \leftarrow (A) - data - (Cy)$	2	12	1
SUBB A, @Ri	A 值减去 Ri 间址单元及 Cy 值, 结果送 A, $A \leftarrow (A) - (Ri) - (Cy)$	1	12	1

② 减 1 指令 (4 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
DEC A	A 值减 1, $A \leftarrow (A) - 1$	1	12	1
DEC Rn	Rn 值减 1, $Rn \leftarrow (Rn) - 1$	1	12	1
DEC direct	Direct 值减 1, $direct \leftarrow (direct) - 1$	2	12	1
DEC @Ri	@Ri 值减 1, $(Ri) \leftarrow (Ri) - 1$	1	12	1

(3) 乘除指令 (2 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MUL AB	A、B 中两无符号数相乘, 结果低 8 位在 A 中, 高 8 位在 B 中, $BA \leftarrow (A) \times (B)$	1	48	4
DIV AB	A、B 中两无符号数相除 (A/B), 结果商送 A, 余数入 B 中。 $A \leftarrow (A) / (B)$ 的商, $B \leftarrow (A) / (B)$ 的余数	1	48	4

(4) 十进制数调整指令

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
DA A	十进制数调整, 对 A 中 BCD 码十进制数加法运算结果进行调整	1	12	1

4. 逻辑运算类指令

逻辑运算指令共有 24 条。分为累加器 A 清零取反 (CLR、CPL) 指令; 与 (ANL) 指令; 或 (ORL) 指令; 异或 (XRL) 指令; 循环移位 (RL、RR、RLC、RRC) 指令。

逻辑运算指令中, 除带进位循环移位指令影响 Cy 和以 PSW (direct) 为目的的操作数的指令外, 其余的逻辑运算指令不影响程序状态 PSW 中的状态标志。

当用逻辑运算指令修改输出口时, 进行的是“读一改一写”操作。

逻辑运算按位进行。

(1) 累加器 A 的清零, 取反指令 (2 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
CLR A	A 值清零, $A \leftarrow 0$	1	12	1
CPL A	A 值按位取反, $A \leftarrow (A)$	1	12	

(2) 逻辑“与”运算指令(6条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
ANL A, Rn	Rn 值和 A 值进行“与”操作, 结果在 A 中, $A \leftarrow (A) \wedge (Rn)$	1	12	1
ANL A, direct	direct 值和 A 值进行“与”操作, 结果在 A 中, $A \leftarrow (A) \wedge (direct)$	2	12	1
ANL A, #data	常数 data 和 A 值进行“与”操作, 结果在 A 中, $A \leftarrow (A) \wedge data$	2	12	1
ANL A, @Ri	@Ri 值和 A 值进行“与”操作, 结果在 A 中, $A \leftarrow (A) \wedge ((Ri))$	1	12	1
ANL direct, A	direct 值和 A 值进行“与”操作, 结果在 direct 中, $direct \leftarrow (A) \wedge (direct)$	2	12	1
ANL direct, #data	direct 值和常数 data 进行“与”操作, 结果在 direct 中, $direct \leftarrow data \wedge (direct)$	3	24	2

(3) 逻辑“或”运算指令(6条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
ORL A, Rn	Rn 值和 A 值进行“或”操作, 结果在 A 中, $A \leftarrow (A) \vee (Rn)$	1	12	1
ORL A, direct	direct 值和 A 值进行“或”操作, 结果在 A 中, $A \leftarrow (A) \vee (direct)$	2	12	1
ORL A, #data	常数 data 和 A 值进行“或”操作, 结果在 A 中, $A \leftarrow (A) \vee data$	2	12	1
ORL A, @Ri	@Ri 值和 A 值进行“或”操作, 结果在 A 中, $A \leftarrow (A) \vee (Ri)$	1	12	1
ORL direct, A	direct 值和 A 值进行“或”操作, 结果在 direct 中, $direct \leftarrow (A) \vee (direct)$	2	12	1
ORL direct, #data	direct 值和常数 data 进行“或”操作, 结果在 direct 中, $direct \leftarrow data \vee (direct)$	3	24	2

(4) 逻辑“异或”运算指令(6条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
XRL A, Rn	Rn 中和 A 值进行“异或”操作, 结果在 A 中, $A \leftarrow (A) \oplus (Rn)$	1	12	1
XRL A, direct	Direct 中和 A 值进行“异或”操作, 结果在 A 中, $A \leftarrow (A) \oplus (direct)$	2	12	1
XRL A, #data	常数 data 和 A 值进行“异或”操作, 结果在 A 中, $A \leftarrow (A) \oplus data$	2	12	1
XRL A, @Ri	@Ri 中和 A 值进行“异或”操作, 结果在 A 中, $A \leftarrow (A) \oplus (Ri)$	1	12	1
XRL direct, A	Direct 中和 A 值进行“异或”操作, 结果在 direct 中, $direct \leftarrow (A) \oplus (direct)$	2	12	1
XRL direct, #data	Direct 值和常数 data 进行“异或”操作, 结果在 direct 中, $direct \leftarrow data \oplus (direct)$	3	24	2

(5) 循环移位指令(4条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
RR A	A 值循环右移(移向低位)一位, A0 移入 A7	1	12	1
RRC A	A 值带进位位循环右移一位, A0 移入 Cy, Cy 移入 A7	1	12	1
RL A	A 值循环左移(移向高位)一位, A7 移入 A0	1	12	1
RLC A	A 值带进位位循环左移一位, A7 移入 Cy, Cy 移入 A0	1	12	1

5. 控制转移指令

控制转移指令共有 22 条。分为无条件转移 (AJMP, LJMP, SJMP, JMP) 指令, 条件转移 (JZ, JNZ, JC, JNC, JB, JNB, JBC, CJNE, DJNZ) 指令, 调用和返回 (ACALL, LCALL, RET, RETI) 指令, 空操作 (NOP) 指令。

控制与转移指令中, 除 “CJNE” 指令对 Cy 有影响外, 其余指令都不影响标志。

控制与转移指令可改变程序计数器 PC 的值, 从而使程序跳到指定的目的地址开始执行。

(1) 无条件转移指令 (4 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
LJMP addr16	长转移: 程序转移到 addr16 指示的地址处, 即 $PC \leftarrow \text{addr16}$	3	24	2
AJMP addr16	绝对转移: 程序转移到 addr16 指示的地址处, 即 $PC \leftarrow PC+2$, PC15—11 不变, $PC10—0 \leftarrow \text{addr10}—0$	2	24	2
SJMP rel	短转移: 程序转移到 rel 指示相对地址处, $PC \leftarrow PC+2$, $PC \leftarrow PC+\text{rel}$	2	24	2
JMP @A+DPTR	间接长转移: 程序转移到 DPTR 为基址加 A 偏移地址处, 即 $PC \leftarrow (A) + (DPTR)$	1	24	2

程序执行无条件转移指令时, 程序就无条件地转移到目的地址。

① 长转移指令: LJMP addr16。

指令的操作是将 16 位目标地址 addr16 装入 PC 中, 允许转移的目标地址在 64KB 空间的任意单元, 用汇编语言编写程序时, addr16 往往是一个标号。

② 绝对转移指令: AJMP addr11。

指令的操作是将 11 位的目标地址 addr11 装入 PC 中的低 11 位。要求目标地址的高 5 位与 PC+2 后 PC 中的高 5 位相同。即转移的目标地址必须和 AJMP 指令的下一条指令首字节地址位于程序存储器的同一段 2KB 范围内, 编写程序时, addr11 也往往是一个标号。

③ 短转移指令: SJMP rel。

指令中相对偏移量 rel 为 8 位的补码, 将其符号扩展为 16 位后与 PC 相加得到 16 位的目标地址。转移的范围为 -128~+127 字节, 编写程序时, rel 同样往往是一个标号。

MCS-51 没有专用的停机指令, 若要动态停机 (原地循环等待) 可以用 SJMP 指令来实现:

动态停机指令:

LP1: SJMP LP1

或写成:

SJMP \$

\$ 表示本指令首字节所在单元的地址, 使用时可省略标号。

④ 间接长转移指令 JMP @A+DPTR。

转移目标地址由数据指针 DPTR 和累加器 A (8 位无符号数) 相加而得。指令的执行不影响累加器 A 和数据指令 DPTR。该指令的特点是转移地址可以在程序运行中加以改变。例如: DPTR 作为基地址, 根据 A 的不同值可以实现多分支转移, 因此一条指令可以完成

多分支转移的功能。该功能称之为散转功能。间接长转移指令又称为散转指令。

(2) 条件转移指令 (13 条)

条件转移指令的操作是判断指定的条件, 如果条件满足则转移, 不满足则顺序执行。

条件转移指令共有 13 条, 可分为判断 A 是否为零转移 (JZ, JNZ) 指令, 位条件转移 (JC, JNC, JB, JNB, JBC) 指令, 比较不等转移 (CJNE) 指令, 减 1 循环 (DJNZ) 转移指令。

① 判断累加器是否为零转移指令 (2 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
JZ rel	A 值为零时, 程序转移到相对地址 rel 处: 若 (A) $\neq 0$, 则 $PC \leftarrow (PC) + 2$; 若 (A) = 0, 则 $PC \leftarrow (PC) + 2 + rel$	2	24	2
JNZ rel	A 值不为零时, 程序转移到相对地址 rel 处: 若 (A) = 0, 则 $PC \leftarrow (PC) + 2$; 若 (A) $\neq 0$, 则 $PC \leftarrow (PC) + 2 + rel$	2	24	2

② 位条件转移指令 (5 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
JC rel	进位位 Cy 为 0 时, 程序转移至 rel: 若 Cy=0, 则 $PC \leftarrow (PC) + 2$; 若 Cy=1, 则 $PC \leftarrow (PC) + 2 + rel$	2	24	2
JNC rel	进位位为 0 时, 程序转移至 rel 处: 若 Cy=0, 则 $PC \leftarrow (PC) + 2 + rel$; 若 Cy=1, 则 $PC \leftarrow (PC) + 2$	2	24	2
JB bit, rel	Bit 位为 1 时, 程序转移至 rel 处: 若 (bit) = 0, 则 $PC \leftarrow (PC) + 3$; 若 (bit) = 1, 则 $PC \leftarrow (PC) + 3 + rel$	3	24	2
JNB bit, rel	Bit 位为 0 时, 程序转移至 rel 处: 若 (bit) = 0, 则 $PC \leftarrow (PC) + 3 + rel$; 若 (bit) = 1, 则 $PC \leftarrow (PC) + 3$	3	24	2
JBC bit, rel	Bit 位为 1 时, 程序转移至 rel 处, 同时将 bit 清零: 若 (bit) = 0, 则 $PC \leftarrow (PC) + 3$; 若 (bit) = 1, 则 (bit) $\leftarrow 0$ 后 $PC \leftarrow (PC) + 3 + rel$	3	24	2

③ 比较不等转移指令 (4 条)。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
CJNE A, #data, rel	#data 与 A 内容不等时转至 rel 处, 同时影响 Cy 位: 若 (A) = data, 则 $PC \leftarrow (PC) + 3$, Cy $\leftarrow 0$; 若 (A) > data, 则 $PC \leftarrow (PC) + 3 + rel$, Cy $\leftarrow 0$; 若 (A) < data, 则 $PC \leftarrow (PC) + 3 + rel$, Cy $\leftarrow 1$	3	24	2
CJNE A, direct, rel	direct 值与 A 值不等时转至 rel 处, 同时影响 Cy 位: 若 (A) = (direct), 则 $PC \leftarrow (PC) + 3$, Cy $\leftarrow 0$; 若 (A) > (direct), 则 $PC \leftarrow (PC) + 3 + rel$, Cy $\leftarrow 0$; 若 (A) < (direct), 则 $PC \leftarrow (PC) + 3 + rel$, Cy $\leftarrow 1$	3	24	2

(续表)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
CJNE Rn, #data, rel	#data 与 Rn 值不等时转至 rel 处, 同时影响 Cy 位: 若 (Rn) = data, 则 $PC \leftarrow (PC) + 3$, $Cy \leftarrow 0$; 若 (Rn) > data 则 $PC \leftarrow (PC) + 3 + rel$, $Cy \leftarrow 0$; 若 (Rn) < data 则 $PC \leftarrow (PC) + 3 + rel$, $Cy \leftarrow 1$	3	24	2
CJNE @Ri, #data, rel	data 与 @Ri 值不等时转至 rel 处, 同时影响 Cy 位: 若 (Ri) = data, 则 $PC \leftarrow (PC) + 3$, $Cy \leftarrow 0$; 若 (Ri) > data, 则 $PC \leftarrow (PC) + 3 + rel$, $Cy \leftarrow 0$; 若 (Ri) < data, 则 $PC \leftarrow (PC) + 3 + rel$, $Cy \leftarrow 1$	3	24	2

④ 减 1 循环指令。

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
DJNZ Rn, rel	Rn 值先减 1, 即 $Rn \leftarrow (Rn) - 1$; 若 Rn 不为零, 则 $PC \leftarrow (PC) + 2 + rel$, 程序转移到相对地址 rel 处; 若 (Rn) = 0, 则 $PC \leftarrow (PC) + 2$, 按原顺序向下执行	2	24	2
DJNZ direct, rel	direct 值先减 1, 即 $direct \leftarrow (direct) - 1$; 若 direct 不为零, 则 $PC \leftarrow (PC) + 3 + rel$, 程序转移到相对地址 rel 处; 若 (direct) = 0, 则 $PC \leftarrow (PC) + 3$, 按原顺序向下执行	3	24	2

(3) 调用和返回指令 (4 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
LCALL addr16	长调用: 程序调用 addr16 处的子程序: $PC \leftarrow (PC) + 3$ $SP \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) 7-0$ $SP \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) 15-8$ $PC \leftarrow addr16$	3	24	2
ACALL addr11	绝对调用: 程序调用 addr11 处的子程序: $PC \leftarrow (PC) + 2$ $SP \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) 7-0$ $SP \leftarrow (SP) + 1$ $(SP) \leftarrow (PC) 15-8$ $PC \leftarrow addr11$	2	24	2
RET	子程序返回: $PC15-8 \leftarrow (SP)$ $SP \leftarrow (SP) - 1$ $PC7-0 \leftarrow (SP)$ $SP \leftarrow (SP) - 1$	1	24	2
RETI	中断返回: $PC15-8 \leftarrow ((SP))$ $SP \leftarrow (SP) - 1$ $PC7-0 \leftarrow (SP)$ $SP \leftarrow (SP) - 1$	1	24	2

(4) 空操作指令 (1 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
NOP	空操作: $PC \leftarrow (PC) + 1$	1	12	1

6. 位操作指令

位操作指令共有 12 条。可分为位传送指令 (MOV)、位状态操作指令 (CLR, CPL, SETB)、位逻辑运算指令 (ANL, ORL)。

(1) 位传送指令 (2 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
MOV C, bit	bit 中状态送入 C 中, $Cy \leftarrow (bit)$	2	12	1
MOV bit, C	C 中状态送入 bit 中, $bit \leftarrow (Cy)$	2	24	2

(2) 位状态操作指令 (6 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
CLR C	Cy 位状态清 0, $Cy \leftarrow 0$	1	12	1
SETB C	Cy 位状态置 1, $Cy \leftarrow 1$	1	12	1
CPL C	Cy 位状态取反, $Cy \leftarrow \overline{Cy}$	1	12	1
CLR bit	bit 位状态清 0, $bit \leftarrow 0$	2	12	1
SETB bit	bit 位状态置 1, $bit \leftarrow 1$	2	12	1
CPL bit	bit 位状态取反, $bit \leftarrow \overline{bit}$	2	12	1

(3) 位逻辑运算指令 (4 条)

汇编指令	操作说明	代码长度 (字节)	指令周期	
			Tosc	TM
ANL C, bit	Bit 中状态和 C 中状态相“与”, 结果送入 Cy, $Cy \leftarrow Cy \wedge (bit)$	2	24	2
ANL C, \overline{bit}	Bit 中状态取反和 C 中状态相“与”, 结果送入 Cy, $Cy \leftarrow (Cy) \wedge (\overline{(bit)})$	2	24	2
ORL C, bit	Bit 中状态和 C 中状态相“或”, 结果送入 Cy, $Cy \leftarrow (Cy) \vee (bit)$	2	24	2
ORL C, \overline{bit}	Bit 中状态取反和 C 中状态相“或”, 结果送入 C, $Cy \leftarrow (Cy) \vee (\overline{(bit)})$	2	24	2

2.1.1.3 汇编语言程序结构

1. 汇编语言常用伪指令

(1) ORG (ORiGin) 汇编起始地址

格式: [〈标号:〉]ORG 〈地址〉

功能: 设置当前段的位置计数器, 如果不用 ORG 规定, 则汇编得到的目标程序将从 0000H 开始。

例: ORG 4000H; 即规定标号 START 地址为 4000H

START: MOV A, #00H; 目标程序的第一条指令从 4000H 开始。

⋮

(2) END (END of assembly) 汇编终止命令

格式: [〈标号:〉]END[〈表达式〉]

功能: 用于终止汇编语言源程序的汇编工作, END 是汇编语言源程序的结束标志, 因此, 在整个汇编语言源程序中只能有一个 END 指令, 且位于程序的最后。如果 END 命令出现在程序中间, 则在 END 之后的指令, 汇编程序将不予处理。

(3) EQU (EQUate) 赋值命令

格式: 〈字符名称〉EQU 〈赋值项〉

其中〈赋值项〉可以是常数、地址、标号或表达式, 其值为 8 位或 16 位二进制数。赋值以后的字符名称既可以做地址使用, 也可以做立即数使用。

功能: 用于给字符名称赋予一个特定值, 赋值以后, 其值在整个程序中有效。

(4) DB (Define Byte) 定义数据字节命令

格式: [〈标号:〉]DB 〈8 位数表〉

功能: 用于从指定的地址单元开始, 在程序存储器的连续单元中定义字节数据, 常用于存放数据表格。

例: 存放 7 段数码管 (共阳极) 显示的十六进制基数 (0~F) 的十六进制数的字形代码, 可使用多条 DB 命令定义。

DB 0C0H, 0F9H, 0A4H, 0B0H; 0, 1, 2, 3

DB 99H, 92H, 82H, 0F8H; 4, 5, 6, 7

DB 80H, 90H, 88H, 83H; 8, 9, A, B

DB 0C6H, 0A1H, 86H, 84H; C, D, E, F

注意

该伪指令只能为程序存储器赋初值, 不能为其他存储器赋初值, 尤其不能为内部数据存储器赋初值。

(5) DW (Define Word) 定义数据字命令

格式: [〈标号:〉]DW 〈16 位数表〉

功能: 用于从指定地址开始, 在程序存储器单元中定义 16 位的字数据。存放时, 数据字的高 8 位在前 (低地址), 低 8 位在后 (高地址)。在 MCS-51 程序设计应用中, 常以 DB 来定义数据, 以 DW 来定义地址。

例: DW “AB”; 存入 41H, 42H。

DB 和 DW 定义的数表, 数的个数不得超过 80 个。如数据的数目较多时, 可使用多个定义命令。

(6) DS (Define Storage) 定义存储区命令

格式: [〈标号:〉]DS 〈16 位数表〉

功能: 用于从指定地址开始, 保留指定数目的字节单元作为存储器, 供程序运行使用,

汇编时，对这些单元不赋值。

(7) BIT 位定义命令

格式：〈字符名称〉 BIT 〈位地址〉

其中〈位地址〉可以是绝对地址，也可以是符号地址（即位符号名称）

功能：用于给字符名称赋以位地址。

例：AQ BIT P1.0

(8) DATA 指令

格式：符号名 DATA 表达式

数值表达式的值在 0~255 之间，表达式必须是一个简单再定位表达式。

功能：用于将一个内部 RAM 的地址赋给指定的符号名。

例：PORT1 DATA 40H

2. 汇编语言程序的基本结构

单片机汇编语言程序设计的基本结构一般分为 4 种形式，即顺序结构、分支结构、循环结构和子程序结构，其基本结构流程如图 2.5 所示。

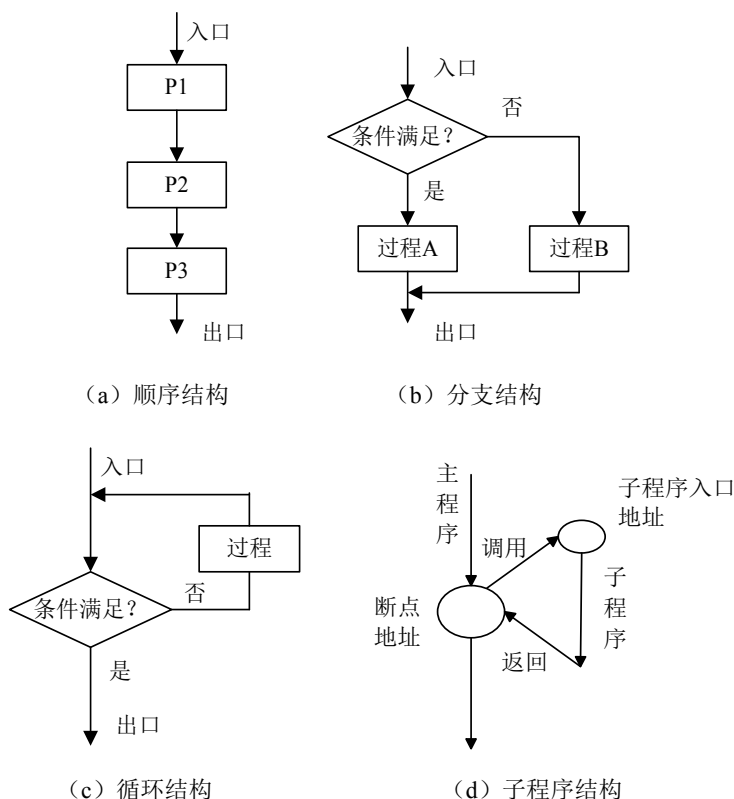


图 2.5 汇编程序基本结构流程

(1) 顺序程序结构

顺序程序是最简单的程序结构，在顺序程序中，既无分支，循环，也不调用子程序，

程序执行时一条一条地按顺序执行指令。

例：设内部 RAM 30H, 31H 单元中分别存放 8 位二进制数，现分别取这两个单元中的半个字节，合并成一个新字节存放在 32H 单元中。要求如下：

32H 单元新字节的低半字节取自 30H 单元的低半字节，而高半字节取自 31H 单元的低半字节。

解：根据题意，给出相应的程序设计流程图如图 2.6 所示。

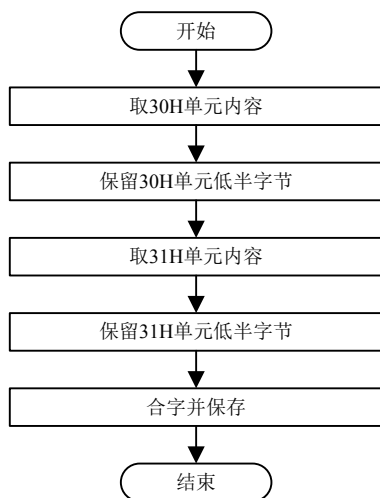


图 2.6 程序设计流程图

程序清单如下：

```

ORG      2000H
START: MOV R1, #30H;  初始化数据指针 R1 的内容
      MOV A, @R1;    取 30H 单元内容送 A
      ANL A, #0FH;    保留 30H 单元内容低 4 位
      INC R1;         修改数据指针 R1 的内容
      XCH A, @R1;     (A) 与 @R1 内容互换
      ANL A, #0FH;    保留 31H 单元内容低四位
      SWAP A;         31H 单元内容高低半字节互换
      ORL A, @R1;     合字生成新字节
      INC R1;         修改数据指针 R1 的内容
      MOV @R1, A;     新字节送 32H 单元保存
      END
  
```

(2) 分支程序结构

程序分支是通过转移指令实现的，也称为选择结构。可分为单分支和多分支两类。

① 单分支程序结构。

单分支程序结构，通过条件转移指令实现，即根据条件对程序的执行进行判断、满足条件则进行程序转移，不满足条件就顺序执行程序。在 MCS-51 指令系统中，可利用 JZ, JNZ, CJNE, DJNZ, JC, JNC, JB, JNB, JBC 等指令，完成为 0、为 1、为正、为负以及相等、不相等等各种条件判断。

例：两个8位无符号二进制数比较大小。假设在外部RAM中有AS1、AS2和AS3共3个连续单元（单元地址从小到大），其中AS1、AS2单元中存放着两个8位无符号二进制数N1，N2，要求找出其中的大数并存入AS3单元中。

解：根据题意，给出相应的程序结构流程图如图2.7所示。

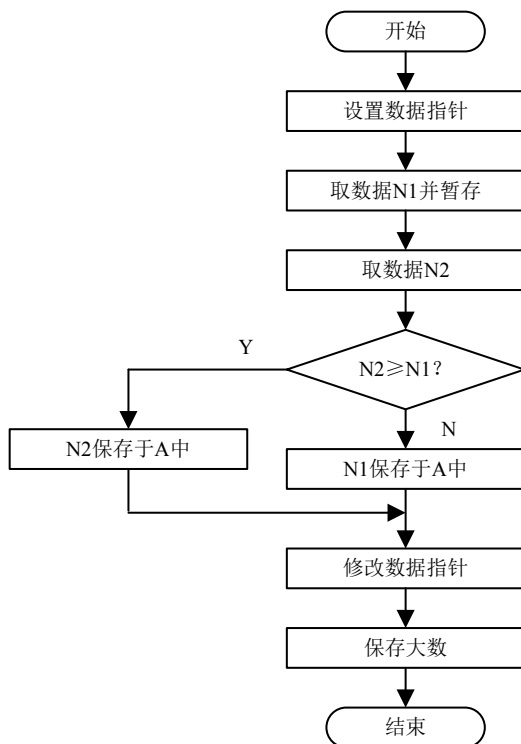


图2.7 单分支程序结构流程图

程序清单如下：

```

ORG 2000H
START: CLR C; 进位清0
MOV DPTR, #AS1; 设数据指针
MOVBX A, @DPTR; A ← ((AS1)), 取N1
MOV R2, A; 暂存N1
INC DPTR; DPTR ← AS2 (指向N2单元)
MOVBX A, @DPTR; 取N2存于A中
SUBB A, R2; N1, N2比较 (N2-N1, 差在A中)
JNC BIG1; N2 ≥ N1, 转BIG1, N2 < N1, 顺序执行
XCH A, R2; N1, N2互换, A ← N1
SJMP BIG0
BIG1: MOVBX A, @DPTR; A ← N2
BIG0: INC DPTR; DPTR ← AS3 (指向N3单元)
MOVBX @DPTR, A; ST3 ← 大数
END
  
```

② 多分支程序结构。

多分支程序是首先把分支程序按序号排列，然后按序号值进行转移。

常使用 **JMP @A+DPTR** 指令：**JMP @A+DPTR** 与数据表配合；**JMP @A+DPTR** 与转移指令配合；利用 **RET** 指令，通过堆栈操作实现。

- **JMP @A+DPTR** 与数据表配合。

数据表存相对地址，范围有限，256 字节。执行完 **JMP** 指令，直接就跳转至分支程序。

例： **MOV A, #n; n 为分支程序序号**

MOV DPTR, #JMPTAB

MOVC A, @A+DPTR

JMP @A+DPTR

JMPTAB: DB ROUT00—JMPTAB

DB ROUT01—JMPTAB

⋮ ⋮

DB ROUTn —JMPTAB

- **JMP @A+DPTR** 与转移指令配合（A 中值为序号与转移指令字节数的乘积）。

例：128 种分支转移程序。

功能：根据入口条件转移到 128 个目的地址。

入口：(R3) = 转移目的地址的序号 00H~7FH。

出口：转移到相应子程序入口。

执行完 **JMP** 指令，程序跳转至分支入口表后再次跳转。

JMP_128: MOV A, R3

RL A

MOV DPTR, #JMPTAB

JMP @A+DPTR

JMPTAB: AJMP ROUT00

AJMP ROUT01

⋮ ⋮

AJMP ROUT7F

说明：此程序要求 128 个转移目的地址（ROUT00 ~ROUT7FH）必须驻留在与绝对转移指令 **AJMP** 相同的一个 2KB 存储区内；**RL** 指令对变址部分乘以 2，因为每条 **AJMP** 指令占两个字节。

- 利用 **RET** 指令，通过堆栈操作实现。

将分支地址从入口地址表取出后，压入堆栈，利用 **RET** 指令赋予 PC 分支地址。

例： **MOV A, R3; 分支程序序号送 A**

RL A; 分支程序序号乘 2

MOV DPTR, #BRTAB; BRTAB 为转移指令表名称标号，

JMP @A+DPTR; 也为转移指令表首地址

BRTAB: AJMP ROUT0; 分支程序 0 的转移指令

AJMP ROUT1; 分支程序 1 的转移指令

```

    AJMP    ROUT2;
    ⋮
    AJMP    ROUT127; 分支程序 127 的转移指令
ROUT0: ⋮⋮⋮; 分支程序 0
ROUT1: ⋮⋮⋮;
    ⋮
ROUT127: ⋮⋮⋮; 分支程序 127

```

(3) 循环程序结构

在程序运行时，有时需要连续重复执行某段程序，可以使用循环程序。MCS-51 汇编语言指令系统没有专用的循环指令，但可以使用条件转移指令通过条件判断来控制循环是继续还是结束。

循环程序一般由四个主要部分组成：

- ① 初始化部分：为循环程序做准备，如规定循环次数、给各变量和地址指针预置初值。
- ② 处理部分：为反复执行的程序段，是循环程序的实体，也是循环程序的主体。
- ③ 循环控制部分：其作用是修改循环变量和控制变量，并判断循环是否结束，直到符合结束条件时，跳出循环为止。
- ④ 结束部分：这部分主要是对循环程序的结果进行分析、处理和存放。

循环程序按结构形式，有单重循环与多重循环。在多重循环中，只允许外重循环嵌套内重循环。不允许循环相互交叉，也不允许从循环程序的外部跳入循环程序的内部，其多重循环示意图如图 2.8 所示。

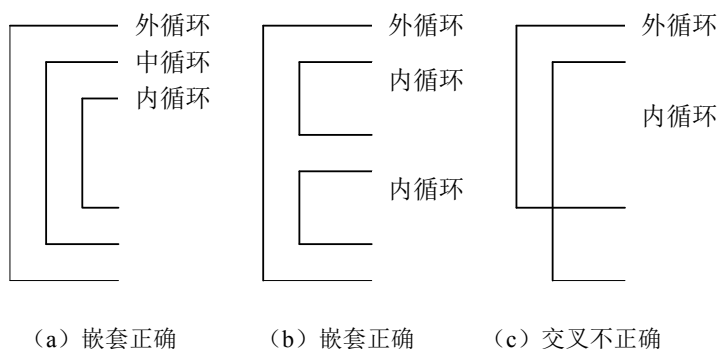


图 2.8 多重循环示意图

例：两个三字节二进制无符号数相加，被加数放在内部 RAM 30H~32H 单元（低字节存放在低地址单元，高字节存放在高地址单元，即低位在前，高位在后），加数放在 3AH~3CH 单元和放在 30H~32H 单元，最高位如有进位，则放在 33H 单元中。

解：根据题意，给出相应的循环程序流程图如图 2.9 所示。

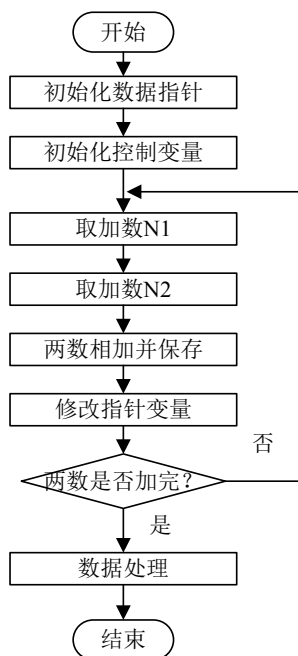


图 2.9 循环程序流程图

程序清单如下：

```

      ORG      2000H
ADDDUO: MOV     R0, #30H; 循环初始化部分
        MOV     R1, #3AH;
        MOV     R7, #03H; 循环次数
        CLR     C;
LOOP:  MOV     A, @R0; 循环体部分
        ADDC    A, @R1;
        MOV     @R0, A;
        INC     R0; 修改指针变量
        INC     R1;
        DJNZ    R7, LOOP; 循环控制部分
        CLR     A; 循环结束处理部分
        ADDC    A, #00H;
        MOV     @R0, A;
        RET;
      END
  
```

(4) 子程序结构

子程序结构是一种非常重要的程序结构。在一个程序中经常遇到反复多次使用某程序段的情况，如果重复书写这个程序段，会使程序变得冗长而杂乱。对此，可采用子程序结构，即把重复的程序段编写为一个子程序，通过主程序调用而使用它。这样不但减少了编程工作量，而且也缩短了程序的长度。

调用和返回构成了子程序调用的完整过程。为了实现这一过程，必须有子程序调用指令和返回指令。调用指令在主程序中使用，而返回指令则应该是子程序的最后一条指令。

执行完这条指令后，程序返回主程序断点处继续执行。

① 子程序的编程原则。

在实际的单片机应用系统软件设计中，为了程序结构更加清晰，易于设计，易于修改，增强程序可读性，基本上都要使用子程序结构。子程序作为一个具有独立功能的程序段，编程时需遵循以下原则：

- 子程序的第一条指令必须有标号，明确子程序入口地址。
- 以返回指令 RET 结束子程序。
- 子程序说明部分。

子程序名称：提供给主程序调用的名字，通常用符号或子程序第一条语句的标号来表示。

子程序功能：简要说明子程序能完成的主要功能。

子程序入口参数：主程序需要向子程序提供的参数。

子程序出口参数：子程序执行完之后向主程序返回的参数。

子程序占用资源：子程序中使用了哪些存储单元、寄存器等。

子程序堆栈深度：子程序占用堆栈区的最大字节数。

子程序嵌套情况：子程序中继续调用子程序的情况。

子程序的字节数：子程序中所有指令字节数的总和。

子程序执行时间：子程序中所有指令的机器周期数总和。

- 较强的通用性和可浮动性，尽可能避免使用具体的内存单元和绝对转移地址等。
- 注意保护现场和恢复现场。

子程序在编制过程中经常会用到一些通用单元，如工作寄存器、累加器、数据指针 DPTR 以及 PSW 等。而这些工作单元在调用它的主程序中也会用到，为此，需要将子程序用到的这些通用编程资源加以保护，称为保护现场。在子程序执行完后需恢复这些单元的内容，称为恢复现场。通常，保护和恢复现场是在子程序中利用堆栈操作实现的，在子程序的开始部分把子程序中要用到的编程资源都保护起来，在执行返回指令之前恢复现场，这是一种比较规范的方法。另外，保护现场和恢复现场也可以在主程序中实现。在调用子程序之前保护现场，子程序返回后恢复现场，这种方式比较灵活，可以根据当时的需要确定要保护的内容。

② 参数传递的方法。

主程序调用子程序时，主程序和子程序之间存在着参数互相传递的问题。参数传递一般有以下几种方法：

- 寄存器传递参数。

通过寄存器 A 传递入口参数和出口参数。

例：假设 a 、 b 均小于 10，计算 $c=a^2+b^2$ ，其中 a 事先存在内部 RAM 的 31H 单元， b 事先存在 32H 单元，请把 c 存入 33H 单元。

```
ORG    2000H;    主程序
MAIN:  MOV    SP, #3FH;  设置栈底
        MOV    A, 31H;    取数 a 存放到 A 中作为入口参数
        LCALL  SQR;
        MOV    R1, A;      出口参数: a 的平方值存放在 A 中
```

```

MOV    A, 32H;    取数  $b$  存放到 A 中作为入口参数
LCALL  SQR;
ADD    A, R1;
MOV    33H, A;
SJMP   $;

```

子程序名称: SQR。

功能: 通过查表求出平方值 $y=x^2$ 。

入口参数: x 存放在累加器 A 中。

出口参数: 求得的平方值 y 存放在 A 中。

占用资源: 累加器 A, 数据指针 DPTR。

```

SQR:   PUSH    DPH; 保护现场, 将主程序中 DPTR 的高 8 位入栈
        PUSH    DPL; 保护现场, 将主程序中 DPTR 的低 8 位入栈
        MOV     DPTR, #TABLE; 在子程序中重新使用 DPTR, DPTR←表首地址
        MOVC    A,    @A+DPTR; 查表
        POP     DPL; 恢复现场, 将主程序中 DPTR 的低 8 位从堆栈中弹出
        POP     DPH; 恢复现场, 将主程序中 DPTR 的高 8 位从堆栈中弹出
        RET
TABLE: DB      0, 1, 4, 9, 16, 25, 36, 49, 64, 81
        END

```

- 利用堆栈传递参数*。

```

        ORG     2000H;    主程序
MAIN:   MOV     SP, #3FH; 设置栈底
        PUSH    31H; 将数  $a$  存放到堆栈中, 作为入口参数
        LCALL  SQR;
        POP     ACC;
        MOV     R1, A; 出口参数:  $a$  的平方值存放在 A 中
        PUSH    ACC;
        LCALL  SQR;
        POP     ACC;
        ADD     A, R1;
        MOV     33H, A;
        SJMP   $;

```

子程序名称: SQR。

功能: 通过查表求出平方值 $y=x^2$ 。

入口参数: x 存放在堆栈中。

出口参数: 求得的平方值 y 存放在堆栈中。

占用资源: 累加器 A, 数据指针 DPTR。

```

SQR:   MOV     R0, SP; R0 作为参数指针
        DEC     R0; 堆栈指针退回子程序调用前的地址

```



```

DEC      R0;
XCH      A, @R0; 保护 ACC, 取出参数
MOV      DPTR, #TABLE; DPTR←表首地址
MOVC     A, @A+DPTR; 查表
XCH      A, @R0; 查表结果放回堆栈中
RET

```

```
TABLE: DB      0, 1, 4, 9, 16, 25, 36, 49, 64, 81
```

③ 子程序调用中应注意的问题

由于子程序调用过程中, CPU 自动使用了堆栈, 因此, 容易出现以下几种错误:

- 忘记给堆栈指针 SP 赋栈底初值, 堆栈初始化位置与第 1 组工作寄存器重合, 如果以不同的方式使用了同一个内存区域, 会导致程序乱套。
- 程序中的 PUSH 和 POP 没有配对使用, 使 RET 指令执行时不能弹出正确的断点地址, 造成返回错误。
- 堆栈设置太小, 堆栈操作增长太大, 使栈区与其他内存单元重合。

2.1.2 C51 程序设计

C51 语言是近年来在国内外 51 单片机开发中普遍使用的一种程序设计语言, C51 能直接对单片机硬件进行操作, 既有高级语言的特点, 又有汇编语言的特点, 因此在单片机应用的程序设计中, 得到非常广泛的应用。

与汇编语言相比, 用 C51 语言进行软件开发, 有如下优点:

(1) 可读性好

C51 语言程序比汇编语言程序的可读性好, 因而编程效率高, 程序便于修改、维护以及程序升级。

(2) 模块化开发与资源共享

C51 开发的模块可直接被其他项目所用, 能很好地利用已有的标准 C 程序资源与丰富的库函数, 减少重复劳动, 也有利于多个工程师的协同开发。

(3) 可移植性好

为某型单片机开发的 C51 程序, 只需将与硬件相关之处和编译链接的参数进行适当修改, 就可方便地移植到其他型号的单片机上。例如, 为 51 单片机编写的程序通过改写头文件以及少量的程序行, 就可以方便地移植到 PIC 单片机上。

(4) 生成的代码效率高

代码效率比直接使用汇编语言低 20% 左右, 如使用优化编译选项, 最高可达 90% 左右, 效果会更好。

C51 与标准 C 语言有许多相同的地方, 但也有自身特点。不同的嵌入式 C 语言编译系统与标准 C 语言的不同, 主要是由于它们所针对的硬件系统不同。对于 51 单片机, 目前广泛使用的是 Keil C51 语言, 简称 C51 语言。

C51 的基本语法与标准 C 相同, C51 在标准 C 的基础上进行了适合于 51 系列单片机硬件的扩展。深入理解 Keil C51 对标准 C 的扩展部分以及不同之处, 是掌握 C51 语言的关键之一。

C51 与标准 C 的主要区别如下:

(1) 库函数不同。

标准 C 中的部分库函数不适合于嵌入式控制器系统,被排除在 Keil C51 之外,如字符屏幕和图像函数。有些库函数可继续使用,但这些库函数都必须针对 51 单片机的硬件特点做出相应的开发。例如库函数 `printf` 和 `scanf`,在标准 C 中,这两个函数通常用于屏幕打印和接收字符,而在 Keil C51 中,主要用于串行口数据的收发。

(2) 数据类型有一定的区别。

在 C51 中增加了几种针对 51 单片机特有的数据类型,在标准 C 的基础上又扩展了 4 种类型。例如,51 单片机包含位操作空间和丰富的位操作指令,因此,C51 语言与标准 C 相比就要增加位类型。

(3) C51 的变量存储模式与标准 C 中的变量存储模式数据不一样。

标准 C 是为通用计算机设计的,计算机中只有一个程序和数据统一寻址的内存空间,而 C51 中变量的存储模式与 51 单片机的存储器紧密相关。

(4) 数据存储类型的不同。

51 单片机存储区可分为内部数据存储区、外部数据存储区以及程序存储区。内部数据存储区可分为 3 个不同的 C51 存储类型: `data`、`idata` 和 `bdata`。外部数据存储区分为 2 个不同的 C51 存储类型: `xdata` 和 `pdata`。在 51 单片机内部或外部,程序存储区只能读不能写。C51 提供了 `code` 存储类型来访问程序存储区。

(5) 标准 C 语言没有处理单片机中断的定义,C51 中有专门的中断函数。

(6) C51 语言与标准 C 语言的输入/输出处理不一样。

C51 语言中的输入/输出是通过 51 单片机的串行口来完成的,输入/输出指令执行前必须对串行口进行初始化。

(7) 头文件的不同。

C51 语言与标准 C 头文件的差异是 C51 头文件必须把 51 单片机内部的外设硬件资源如定时器、中断、I/O 等所相应的功能寄存器写入头文件内。

(8) 程序结构的差异。

由于 51 单片机硬件资源有限,它的编译系统不允许太多的程序嵌套。其次,标准 C 所具备的递归特性不被 C51 语言支持。

但是从数据运算操作、程序控制语句以及函数的使用上来说,Keil C51 与标准 C 几乎没有什么明显的差别。

如果程序设计者具备了有关标准 C 的编程基础,只要注意 Keil C51 与标准 C 的不同之处,并熟悉 51 单片机的硬件结构,就能够较快地掌握 C51 的编程。在标准 C 的基础上了解掌握 C51 的数据类型和存储类型、基本运算与流程控制语句、C51 语言构造数据类型、C51 函数以及 C51 程序设计的其他问题,为 C51 程序设计打下基础。

2.1.2.1 C51 语言中的数据类型与存储类型

1. 数据类型

数据是单片机操作的对象,是具有一定格式的数字或数值,数据的不同格式就称为数

据类型。Keil C51 支持的基本数据类型如表 2.1 所示。针对 51 单片机的硬件特点，C51 在标准 C 的基础上，扩展了 4 种数据类型（表 2.1 中最后 4 行）。

注意

扩展的 4 种数据类型，不能使用指针对它们存取。

表 2.1 Keil C51 支持的数据类型

数据类型	位 数	字 节 数	取值范围
signed char	8	1	-128~+127，有符号字符变量
unsigned char	8	1	0~255，无符号字符变量
Enum	16	2	-32768~+32767
signed short	16	2	-32768~+32767
unsigned short	16	2	0~65535
signed int	16	2	-32768~+32767
unsigned int	16	2	0~65535
signed long	32	4	-2147483648~+2147483647
unsigned long	32	4	0~4294967295
float	32	4	$\pm 1.175494\text{E}-38 \sim \pm 3.402823\text{E}+38$
bit	1		0~1
sbit	1		0~1
sfr	8	1	0~255
sfr16	16	2	0~65535

2. C51 的扩展数据类型

(1) 位变量 bit

bit 的值可以是 1（true），也可以是 0（false）。

(2) 特殊功能寄存器 sfr

特殊功能寄存器分布在片内数据存储区的地址单元 80H~FFH 之间，“sfr”数据类型占用一个内存单元。利用它可以访问 51 单片机内部的所有特殊功能寄存器。例如：sfr P1=0x90 这一语句定义了 P1 端口在片内的寄存器，在程序后续的语句中可以用“P1=0xff”使 P1 的所有引脚输出为高电平之类的语句来操作特殊功能寄存器。

(3) 特殊功能寄存器 sfr16

“sfr16”数据类型占两个内存单元。它用于操作占两个字节的特殊功能寄存器。

例如：“sfr16 DPTR=0x82”语句定义了片内 16 位数据指针寄存器 DPTR，其低 8 位字节地址为 82H，高 8 位字节地址为 83H。

(4) 特殊功能位 sbit

sbit—片内特殊功能寄存器的可寻址位。例如：

```
sfr    PSW=0xd0;          /*定义 PSW 寄存器地址为 0xd0*/
sbit   PSW^2 = 0xd2;      /*定义 OV 位为 PSW.2*/
```

符号“^”前是特殊功能寄存器的名字，“^”的后面数字是特殊功能寄存器可寻址位在寄存器中的位置，取值必须是0~7。

注意

不要把 bit 与 sbit 混淆。bit 是定义普通的位变量，值只能是二进制数的 0 或 1。而 sbit 定义的是特殊功能寄存器的可寻址位，它的值是可进行位寻址的特殊功能寄存器的某位的绝对地址，例如，PSW 寄存器 OV 位的绝对地址 0xd2。

3. 数据存储类型

在讨论 C51 的数据类型时，必须同时提及它的存储类型，以及它与 51 单片机存储器结构的关系，因为 C51 定义的任何数据类型必须以一定的方式定位在 51 单片机的某一存储区中，否则没有任何实际意义。

51 单片机有片内、外数据存储区，还有程序存储区。51 单片机片内的数据存储区是可读写的，51 单片机的衍生系列最多可有 256 个字节的内部数据存储区，其中低 128 字节可直接寻址，高 128 字节（80H~FFH）只能直接寻址，从 20H 开始的 16 字节可位寻址。内部数据存储区可分为 3 个不同的数据存储类型：data、idata 和 bdata。

访问片外数据存储区比访问片内数据存储区慢，因为片外数据存储区是通过数据指针加载地址来间接寻址访问的。C51 提供两种不同数据存储类型 xdata 和 pdata 来访问片外数据存储区。

程序存储区只能读不能写，可能在 51 单片机内部或者外部，或者外部和内部都有，由 51 单片机的硬件决定，C51 提供了 code 存储类型来访问程序存储区。

C51 存储类型与 51 单片机实际的存储空间的对对应关系如表 2.2 所示。

表 2.2 C51 的存储器类型

存储器类型	长度（位）	存储位置	说 明
data	8	片内 RAM	直接寻址：00~7FH，速度最快
bdata	1		可位寻址：00H~7FH
idata	8		间接寻址：00~FFH
pdata	8	片外 RAM	分页间址：00~FFH
xdata	16		间接寻址：0000~FFFFH
code	16	ROM	间接寻址：0000~FFFFH

单片机访问片内 RAM 比访问片外 RAM 相对快一些，所以应当尽量把频繁使用的变量置于片内 RAM。即采用 data、bdata 或 idata 存储类型，而将容量较大的或使用不太频繁的那些变量置于片外 RAM，即采用 pdata 或 xdata 存储类型。常量只能采用 code 存储类型。

变量存储类型定义举例：

(1) char data a1; /*字符变量 a1 被定义为 data 型，分配在片内 RAM 低 128 字节中*/

(2) float idata x,y; /*浮点型变量 x 和 y 被定义为 idata 型，定位在片内 RAM 中，只能用间接寻址方式寻址*/

(3) `bit bdata p; /*位变量 p 被定义为 bdata 型, 定位在片内 RAM 中的位寻址区*/`

(4) `unsigned int pdata var1; /*无符号整型变量 var1 被定义为 pdata 型, 定位在片外 RAM 中, 相当于使用 @Ri 间接寻址*/`

(5) `unsigned char xdata a[2][4]; /*无符号字符型二维数组变量 a[2][4] 被定义为 xdata 存储类型, 定位在片外 RAM 中, 占据 2×4=8 个字节, 相当于使用 @DPTR 间接寻址*/`

4. 数据存储模式

如在变量定义时略去存储类型标识符, 编译器会自动默认存储类型, 如表 2.3 所示。

表 2.3 默认存储器类型

存储模式	默认存储类型	特点说明
SMALL	data (小模式)	存储于片内 RAM, 速度快
COMPACT	pdata (紧凑模式)	存储于片外分页 RAM
LARGE	xdata (大模式)	存储于片外 64K 的 RAM, 速度慢

2.1.2.2 C51 的特殊功能寄存器及位变量定义

1. 特殊功能寄存器的 C51 定义

C51 语言允许使用关键字 `sfr`、`sbit` 或直接引用编译器提供的头文件来对特殊功能寄存器 (SFR) 进行访问, 特殊功能寄存器在片内 RAM 的高 128 字节, 只能采用直接寻址方式。

(1) 使用关键字定义 `sfr`

为了能直接访问特殊功能寄存器 SFR, C51 语言提供了一种定义方法, 即引入关键字 `sfr`, 语法如下:

`sfr 特殊功能寄存器名字=特殊功能寄存器地址;`

例如:

`sfr IE=0xA8; /*中断允许寄存器地址 A8H*/`

要访问 16 位 SFR, 可使用关键字 `sfr16`。16 位 SFR 的低字节地址必须作为 “`sfr16`” 的定义地址, 例如:

`sfr16 DPTR=0x82; /*数据指针 DPTR 的低 8 位地址为 82H, 高 8 位地址为 83H*/`

(2) 通过头文件访问 SFR

各种衍生的 51 单片机的特殊功能寄存器的数量与类型有时是不相同的, 对单片机特殊功能寄存器的访问可以通过头文件的访问来进行。

为了用户处理方便, C51 语言把 51 单片机 (或 52 单片机) 的常用的特殊功能寄存器和其中的可寻址位进行了定义, 放在一个 `reg51.h` (或 `reg52.h`) 的头文件中。当用户要使用时, 只需在使用之前用一条预处理命令 `#include<reg51.h>` 把这个头文件包含到程序中, 就可以使用特殊功能寄存器名和其中的可寻址位名称了。用户可以通过文本编辑器对头文件进行增减。

头文件引用举例如下:

`#include<reg51.h> /*头文件为 51 型单片机的头文件*/`

```

void main(void)
{
    TL0=0xF0;      /*给定时器 T0 低字节 TL0 设置时间常数，已在 reg51.h 中定义*/
    TH0=0x3F;      /*给 T0 高字节 TH0 设时间常数*/
    TR0=1;         /*启动定时器 0 */
    .....
}

```

(3) 特殊功能寄存器中的位定义

对 SFR 中的可寻址位的访问，要使用关键字来定义可寻址位，共有 3 种方法。

① sbit 位名=特殊功能寄存器^位置。

例如：

```

sfr    PSW=0xD0;      /*定义 PSW 寄存器的字节地址 0xD0H*/
sbit   CY=PSW^7;      /*定义 CY 位为 PSW.7，地址为 0xD0*/
sbit   OV=PSW^2;      /*定义 OV 位为 PSW.2，地址为 0xD2*/

```

② sbit 位名=字节地址^位置。

例如：

```

sbit   CY=0xD0^7; /* CY 位地址为 0xD7*/
sbit   OV=0xD0^2; /* OV 位地址为 0xD2*/

```

③ sbit 位名=位地址。

这种方法将位的绝对地址赋给变量，位地址必须在 0x80~0xFF 之间。

例如：

```

sbit   CY=0xD7; /* CY 位地址为 0xD7*/
sbit   OV=0xD2; /* OV 位地址为 0xD2*/

```

2. 位变量的 C51

(1) 位变量的 C51 定义

由于 51 单片机能够进行位操作，C51 扩展的“bit”数据类型用来定义位变量，这是 C51 与标准 C 的不同之处。

C51 采用关键字“bit”来定义位变量，一般格式为：

```
bit bit_name;
```

例如：

```
bit ov_flag;          /* 将 ov_flag 定义为位变量*/
```

(2) 函数可以包含类型为 bit 的参数，也可将其作为返回值

C51 程序函数可以包含类型为“bit”的参数，也可将其作为返回值。例如：

```

bit func(bit b0, bit b1); /* 位变量 b0 与 b1 作为函数 func 的参数*/
{
    .....
return(b1);          /* 位变量 b1 作为函数的返回值*/
}

```

(3) 位变量定义的限制

位变量不能用来定义指针和数组。例如：

```
bit *ptr;          /* 错误，不能用位变量来定义指针*/
```

```
bit array[] ; /* 错误，不能用位变量来定义数组 array[]*/
```

在定义位变量时，允许定义存储类型，位变量都被放入一个位段，此段总是位于 51 单片机的片内 RAM 中，因此其存储类型限制为 data 或 idata，如果将位变量定义成其他类型都会导致编译时出错。

2.1.2.3 C51 流程控制语句

1. if 语句

选择语句 if 的基本结构是：if(表达式) {语句;}

如果括号中的表达式成立（为真），则程序执行花括号中的语句；否则程序将跳过花括号中的语句部分，执行下面其他语句。其他形式的 if 语句：

形式一： if(表达式) {语句; }

形式二： if(表达式) {语句 1; } else {语句 2; }

形式三： if(表达式 1) {语句 1; }
 else if(表达式 2) {语句 2; }
 else if(表达式 3) {语句 3; }
 else {语句 4; }

2. switch/case 语句

switch/case 语句的基本结构是：

```
switch (表达式)
{
    case 常量表达式 1 : {语句 1;} break;
    case 常量表达式 2 : {语句 2;} break;
    case 常量表达式 3 : {语句 3;} break;
    default      : {语句 4;}
}
```

当 switch 括号中表达式的值与某一 case 后面常量表达式的值相等时，就执行它后面的语句，然后遇到 break 而退出 switch 语句。当所有的 case 中的常量表达式的值都没有与表达式的值匹配时，就执行 default 后面的语句。

【例】在单片机程序设计中，常用 switch 语句作为键盘中按键按下的判别，并根据按下键的键号跳向各自的分支处理程序。

```
keynum=keyscan();
switch (keynum)
{
    case 1:key1();break;          /*如果按下键的键值为 1，则执行函数 key1()*/
    case 2:key2(); break;        /*如果按下键的键值为 2，则执行函数 key2()*/
    case 3:key3(); break;        /*如果按下键的键值为 3，则执行函数 key3()*/
```

```
case 4:key4( ); break;          /*如果按下键的键值为 4，则执行函数 key4( )*/
.....
default:
}
```

3. while 语句……循环

(1) while do 语句:

while (表达式)

[do]{语句; } /*循环体*/

表达式是能否循环的条件，语句是循环体。当表达式为非 0（真）时，则重复执行循环体内的语句；当表达式为 0（假）时，则中止 while 循环，程序将执行循环结构之外的下一条语句。它的特点是：先判断条件，后执行循环体。在循环体中对条件进行改变，然后再判断条件，如条件成立，则再执行循环体，如条件不成立，则退出循环。如条件第一次就不成立，则循环体一次也不执行。

例如：

while ((P1&0x80) == 0)

{ }

while 中的条件语句对 51 单片机的 P1 口 P1.7 进行测试，如果 P1.7 为低电平（0），则由于循环体无实际操作语句，故继续测试下去（等待），一旦 P1.7 的电平变高（1），则循环终止。

(2) do while 语句:

do

{语句; } /*循环体*/

while (表达式);

执行过程：先执行循环体中的语句，后判断表达式。如表达式成立（真），则再执行循环体，然后又判断，直到有表达式不成立（假）时，退出循环，执行 do while 结构的下一条语句。do while 语句在执行时，循环体内的语句至少会被执行一次。

4. for 语句……循环

for 语句是 C51 中使用最灵活、最频繁的循环控制语句，可以完全代替 while 语句，功能最强大，格式如下：

for (表达式 1; 表达式 2; 表达式 3)

{语句; } /*循环体*/

for 语句后面带三个表达式，它的执行过程如下：

(1) 先求解表达式 1 的值。

(2) 求解表达式 2 的值，如果为真，则执行循环体语句，然后执行表达式 3，执行完毕接着判断表达式 2 的值，以此类推；如果为假，则 for 循环结束。

在 for 循环中，一般表达式 1 为初值表达式，用于给循环变量赋初值；表达式 2 为条件表达式，对循环变量进行判断；表达式 3 为循环变量更新表达式，用于对循环变量的值进行更新，使循环变量能不满足条件而退出循环。

例如：编写一个延时 1ms 程序。

```
void delays( unsigned char int j)
{
    unsigned char i;
    while(j--)
    {
        for(i=0;i<125;i++);
    }
}
```

如果把上述程序段编译成汇编语言代码进行分析,用 for 进行的内部循环大约延时 8 ms,但不是特别精确。不同的编译器会产生不同的延时,因此 *i* 的上限值 125 应根据实际情况进行补偿调整。

5. break 和 continue 语句

break 和 continue 语句通常用于循环结构中,用来跳出循环结构。但是二者又有所不同,下面分别介绍。

(1) break 语句

break 语句可以跳出 switch 结构,使程序继续执行 switch 结构后面的一个语句。

break 语句可以从循环体中跳出,提前结束循环而接着执行循环结构外的下一条语句。

break 语句只能用在循环语句和 switch 语句之中。

(2) continue 语句

continue 语句用在循环结构中,用于结束本次循环,跳过循环体中 continue 下面尚未执行的语句,直接进行下一次是否执行循环的判定。

continue 语句和 break 语句的区别在于:continue 语句只是结束本次循环而不是终止整个循环;break 语句则是结束循环,不再进行条件判断。

6. return 语句

return 语句一般放在函数的最后位置,用于终止函数的执行,并控制程序返回调用该函数时所处的位置。返回时还可以通过 return 语句带回返回值。return 语句格式有两种:

(1) return;

(2) return (表达式);

如果 return 语句后面带有表达式,则要计算表达式的值,并将表达式的值作为函数的返回值。若不带表达式,则函数返回时将返回一个不确定的值。通常用 return 语句把调用函数取得的值返回给主调用函数。

2.1.2.4 C51 语言的函数

C51 语言中函数的数目是不限制的,但是一个 C51 程序必须至少有一个函数,以 main 为名,称为主函数,主函数是唯一的,整个程序从这个主函数开始执行。

C51 语言还可建立和使用库函数,可由用户根据需求调用。

1. 函数的分类

从结构上分，C51 语言函数可分为主函数 `main()` 和普通函数两种。而普通函数又划分为两种：标准库函数和用户自定义函数。

(1) 标准库函数

标准库函数是由 C51 编译器提供的。编程者在进行程序设计时，应该善于充分利用这些功能强大、资源丰富的标准库函数资源，以提高编程效率。

用户可直接调用 C51 库函数而不需为这个函数写任何代码，只需要包含具有该函数说明的头文件即可。例如调用输出函数 `printf` 时，要求程序在调用输出库函数前包含以下的 `include` 命令：

```
#include <stdio.h>
```

(2) 用户自定义函数

用户自定义函数是用户根据需要所编写的函数。从函数定义的形式分为：无参函数、有参函数和空函数。

① 无参函数。

此种函数在被调用时，既无参数输入，也不返回结果给调用函数，只是为完成某种操作而编写的函数。

无参函数的定义形式为：

返回值类型标识符 函数名()

```
{
    函数体;
}
```

无参函数一般不带返回值，因此函数的返回值类型的标识符可省略。

例如函数：`main()`，该函数为无参函数，返回值类型的标识符可省略，默认值是 `int` 类型。

② 有参函数。

调用此种函数时，必须提供实际的输入参数。有参函数的定义形式为：

返回值类型标识符，函数名（形式参数列表）。

形式参数说明

```
{
    函数体;
}
```

【例】定义一个函数 `max()`，用于求两个数中的大数。

```
int a,b;
int max(a, b)
{
    if (a>b) return (a) ;
    else return (b) ;
}
```

上面程序段中，`a`、`b` 为形式参数。`return()` 为返回语句。

③ 空函数。

此种函数体内是空白的。调用空函数时，什么工作也不做，不起任何作用。定义空函数的目的，并不是为了执行某种操作，而是为了以后程序功能的扩充。先将一些基本模块的功能函数定义成空函数，占好位置，并写好注释，以后再用一个编好的函数代替它。这样整个程序的结构清晰，可读性好，以后扩充新功能方便。

空函数的定义形式为：

返回值类型标识符 函数名()

{ }

例如：

float min()

{ } /*空函数，占好位置*/

2. 函数的参数与返回值

(1) 函数的参数

C语言采用函数之间的参数传递方式，使一个函数能对不同的变量进行功能相同的处理，从而大大提高了函数的通用性与灵活性。

函数之间的参数传递，由主函数调用时主调函数的实际参数与被调函数的形式参数之间进行数据传递来实现。

被调用函数的最后结果由被调用函数的 `return` 语句返回给调用函数。

函数的参数包括形式参数和实际参数。

① 形式参数：函数的函数名后面括号中的变量名称为形式参数，简称形参。

② 实际参数：在函数调用时，主调函数名后面括号中的表达式称实际参数，简称实参。

在C语言的函数调用中，实际参数与形式参数之间的数据传递是单向进行的，只能由实际参数传递给形式参数，而不能由形式参数传递给实际参数。

实际参数与形式参数的类型必须一致，否则会发生类型不匹配的错误。被调用函数的形式参数在函数未调用之前，并不占用实际内存单元。只有当函数调用发生时，被调用函数的形式参数才分配给内存单元，此时内存中调用函数的实际参数和被调用函数的形式参数位于不同的单元。在调用结束后，形式参数所占有的内存被系统释放，而实际参数所占有的内存单元仍保留并维持原值。

(2) 函数的返回值

函数的返回值是通过函数中的 `return` 语句获得的。一个函数可以有一个以上的 `return` 语句，但是多于一个的 `return` 语句必须在选择结构（`if` 或 `do/case`）中使用（例如前面求两个数中的大数函数 `max()` 的例子），因为被调用函数一定只能返回一个变量。

函数返回值的类型一般在定义函数时，由返回值的标识符来指定。例如在函数名之前的 `int` 指定函数的返回值的类型为整型数（`int`）。若没有指定函数的返回值类型，默认返回值为整型类型。

当函数没有返回值时，则使用标识符 `void` 进行说明。

3. 函数的调用

在一个函数中需要用到某个函数的功能时，就调用该函数。调用者称为主调函数，被

调用者称为被调函数。

(1) 函数调用的一般形式

函数调用的一般形式：

函数名 (实际参数列表)；

若被调函数是有参函数，则主调函数必须把被调函数所需的参数传递给被调函数。传递给被调函数的数据称为实际参数（简称实参），必须与形参的数据在数量、类型和顺序上都一致。实参可以是常量、变量和表达式。实参对形参的数据是单向的，即只能将实参传递给形参。

(2) 函数调用的方式

主调用函数对被调用函数的调用有以下 3 种方式。

① 函数调用语句。

函数调用语句把被调用函数的函数名作为主调函数的一个语句。例如：

```
message( );
```

此时，并不要求函数返回结果数值，只要求函数完成某种操作。

② 函数参数。

函数参数即被调用函数作为另一个函数的实际参数，例如：

```
m=max(a,gcd(u,v));
```

其中，gcd(u,v)是一次函数调用，它的值作为另一个函数的 max() 的实际参数之一。

(3) 对调用函数的说明

在一个函数调用另一个函数时，需具备以下条件：

① 被调用函数必须是已经存在的函数（库函数或用户自定义的函数）。

② 如果程序中使用了库函数，或使用了不在同一文件中的自定义函数，则应该在程序的开头处使用#include 包含语句，将所有的函数信息包含到程序中来。

③ 如果程序中使用了自定义函数，且该函数与调用它的函数同在一个文件中，则应根据主调用函数与被调用函数在文件中的位置，决定是否对被调用函数作出说明。

如果被调用函数在主调用函数之后，一般应在主调用函数中，在被调用函数调用之前，对被调用函数的返回值类型作出说明。

如果被调用函数出现在主调用函数之前，不用对被调用函数进行说明。

如果在所有函数定义之前，在文件的开头处，在函数的外部已经说明了函数的类型，则在主调用函数中不必对所调用的函数再做返回值类型说明。

4. 中断服务函数

由于标准 C 没有处理单片机中断的定义，为了能进行 51 单片机的中断处理，C51 编译器对函数的定义进行了扩展，增加了一个扩展关键字 interrupt。使用 interrupt 可以将一个函数定义成中断服务函数。由于 C51 编译器在编译时对声明为中断服务程序的函数自动添加了相应的现场保护、阻断其他中断、返回时自动恢复现场等处理的程序段，因而在编写中断服务函数时可不考虑这些问题，减小了用户编写中断服务程序的繁琐程度。

中断服务函数的一般形式为：

函数类型 函数名 (形式参数表) interrupt n [using m]

关键字 `interrupt` 后面的 n 是中断号, n 的取值范围为 $0\sim 31$ 。编译器从 $8n+3$ 处产生中断向量, 具体的中断号 n 和中断向量取决于 8051 系列单片机芯片的型号, 常用中断源和中断向量如表 2.4 所示。

表 2.4 单片机常用中断号与中断向量

中断号 n	中 断 源	中断向量 $8n+3$
0	外部中断 0	0003H
1	定时器/计数器 0	000BH
2	外部中断 1	0013H
3	定时器/计数器 1	001BH
4	串行口	0023H

关键字 `using` 后的 m 是所选择的寄存器组, `using` 是一个选项, 可省略。如果没有使用 `using` 关键字指明寄存器组, 中断函数中的所有工作寄存器的内容将被保存到堆栈中。

一个函数定义为中断服务函数后, 编译器自动生成中断向量和程序的入栈及出栈代码, 从而提高了工作的效率。

使用中断服务函数时应注意:

- (1) 中断函数不能直接调用中断函数。
- (2) 不能通过形式参数来进行参数传递。
- (3) 在中断函数中调用其他函数, 两者所使用的寄存器组应相同。
- (4) 中断函数没有返回值。
- (5) 关键字 `interrupt` 和 `using` 的后面都不允许使用带运算符的表达式。

5. C51 库函数

C51 编译器的运行库中包含有丰富的库函数, 使用库函数可大大简化程序设计工作, 提高工作效率。下面对这些库函数进行分类说明。

(1) 一般 I/O 函数 STDIO.H

C51 库中含有字符 I/O 函数, 它们通过系列单片机的串行接口工作, 如果希望支持其他 I/O 接口, 只要改动 `getkey()` 和 `putchar()` 函数, 库中所有其他 I/O 支持函数都依赖于这两个函数模块, 不需要改动。另外需要注意, 在使用 51 系列单片机的串行口之前, 应先进行初始化。例如, 以 2400 波特率 (12 MHz 时钟频率) 初始化串行口程序如下:

```
SCON=0x52;
```

```
TMOD=0x20;
```

```
TH1=0x53;
```

```
TR1=1;
```

详细库函数列表如表 2.5 所示。

表 2.5 I/O 函数 STDIO.H

函 数	功 能
<code>extern char _getkey (void)</code>	从串口中读入一个字符, 不显示
<code>extern char getchar (void)</code>	从串口读入一个字符, 并通过串口输出对应的字符

(续表)

函 数	功 能
extern char ungetchar (char)	将输入的字符回送输入缓冲区,因此下次 gets 或者 getchar 可以使用该字符。成功时返回“char”,失败时返回 EOF,不能用 ungetchar 处理多个字符
extern char putchar (char)	通过 51 单片机的串口输出字符,与 _getkey()函数一样
extern char puts (const char *)	将字符串和换行符写入串行口,错误时返回 EOF,否则返回一个非负数
extern char gets (char *s, int n)	该函数通过 getchar()函数从串口中读入一个长度为 n 的字符串并存入由“s”指向的数组。输入时一旦检测到换行符就结束字符输入。输入成功时返回传入的参数指针,失败时返回 NULL
extern int printf (const char *,...)	printf 以一定的格式通过 8051 串行口输出数值和字符串,返回值为实际输出的字符数。参数可以是字符串指针、字符和数值
extern int sprintf (char *,const char *, ...)	与 printf 功能相似,但数据不是输出到串行口,而是通过一个指针 s 送入可寻址的内存缓冲区,并以 ASCII 码的形式储存。该函数允许的输出参数总字节数与 printf 完全相同
extern int vprintf (const char *,char*)	与 printf 的功能基本相同,但是该函数使用指针来作为参数列表
extern int vsprintf (char *,const char *,char*)	与 printf 的功能基本相同,但是该函数使用指针来作为参数列表
extern char *gets (char *s, int n)	该函数通过 getchar()函数从串口中读入一个长度为 n 的字符串并存入由“s”指向的数组。输入时一旦检测到换行符就结束字符输入。输入成功时返回传入的参数指针,失败时返回 NULL
extern int scanf (const char *,...)	该函数在格式控制串的控制下,利用 getchar 函数从串行口中读入数据,每遇到一个符合格式控制串规定的值,就将它按顺序存入由参数指针 指向的存储单元。注意,每个参数必须是指针。scanf 返回它所发现并转换的输入项数,若遇到错误则返回 EOF
extern int sscanf (char *,const char *,...)	该函数与 scanf 的输入方式相似,但是字符串的输入不是通过串行口,而是通过另一个以空结束的指针。sscanf 参数允许的总字节数受 C51 库的限制,在 SMALL 和 COMPACT 编译模式下,最大允许传递 40Byte 的参数
extern int puts (const char *)	将字符串和换行符写入串行口,错误时返回 EOF,否则返回一个非负数

(2) 数学函数 MATH.H

数学相关的函数如表 2.6 所示。

表 2.6 数学函数 MATH.H

函 数	功 能
int abs (int val)	计算并返回 val 的绝对值,变量和返回值类型都是整型的
float abs (float val)	计算并返回 val 的绝对值,变量和返回值类型都是单精度型的
char abs (char val)	计算并返回 val 的绝对值,变量和返回值类型都是字符型的
long abs (long val)	计算并返回 val 的绝对值,变量和返回值类型都是长整型的
float exp (float x)	返回以 e 为底 x 的幂
float log (float x)	返回自然对数
float log10 (float x)	返回以 10 为底的对数
float sqrt (float x)	返回 x 的正平方根
float cos (float x)	返回相应的余弦值。变量范围必须在 $-\frac{\pi}{2} \sim +\frac{\pi}{2}$ 之间,否则会返回错误
float sin (float x)	返回相应的正弦值。变量范围必须在 $-\frac{\pi}{2} \sim +\frac{\pi}{2}$ 之间,否则会返回错误

(续表)

函 数	功 能
float tan (float x)	返回相应的正切值。变量范围必须在 $-\frac{\pi}{2} \sim +\frac{\pi}{2}$ 之间, 否则会返回错误
float acos (float x)	返回 x 的反余弦值
float asin (float x)	返回 x 的反正弦值
float atan (float x)	返回 x 的反正切值, 值域为 $-\frac{\pi}{2} \sim +\frac{\pi}{2}$
float atan2 (float x)	atan2 返回 x/y 的反正切值, 值域为 $-\pi \sim +\pi$
float cosh (float x)	返回 x 的相应的双曲函数值
float sinh (float x)	返回 x 的相应的双曲函数值
float tanh (float x)	返回 x 的相应的双曲函数值
void fpsave (struct FPBUF *p)	保存浮点字程序的状态, 当中断程序中需要执行浮点运算的时候很有用
void fpstore (struct FPBUF *p)	恢复浮点字程序的原始状态, 当中断程序中需要执行浮点运算的时候很有用
float ceil (float x)	ceil 返回一个不小于 x 的最小整数 (作为浮点数)
float floor (float x)	返回一个不大于 x 的最大整数 (作为浮点数)
float modf (float x, float *ip)	将浮点数 x 分为整数和小数两个部分, 两者都含有与 x 相同的符号, 整数部分 *ip, 小数部分作为返回值
float pow (float x, float y)	计算 x^y

(3) 字符函数 CTYPE.H

字符函数主要用来判断是哪种类型的字符, 具体如表 2.7 所列。

表 2.7 字符函数 CTYPE.H

函 数	功 能
extern bit isalpha (unsigned char)	检查参数字符是否为英文字符, 是则返回 1, 否则返回 0
extern bit isalnum (unsigned char)	检查参数字符是否为字母或者数字字符, 是则返回 1, 否则返回 0
extern bit iscntrl (unsigned char)	检查参数字符是否在 0x00~0x1F 之间或者等于 0x7F, 是则返回 1, 否则返回 0
extern bit isdigit (unsigned char)	检查参数字符是否为数字字符, 是则返回 1, 否则返回 0
extern bit isgraph (unsigned char)	检查参数字符是否为打印字符, 可打印字符的值域为 0x21~0x7E。为真时返回值为 1, 否则返回 0
extern bit isprint (unsigned char)	除了接受空格符 (0x20) 以外, 其余与 isgraph 相同
extern bit ispunct (unsigned char)	检查参数字符是否为标点、空格或者格式字符。如果是空格、32 个标点和格式字符则返回 1, 否则返回 0
extern bit islower (unsigned char)	检查参数字符是否为小写字母, 是则返回 1, 否则返回 0
extern bit isupper (unsigned char)	检查参数字符是否为大写字符, 是则返回 1, 否则返回 0
extern bit isspace (unsigned char)	检查参数字符是否为下列之一: 空格、制表符、回车、换行、垂直制表符和送纸符号, 是则返回 1, 否则返回 0
extern bit isxdigit (unsigned char)	检查参数字符是否为十六进制数字字符, 是则返回 1, 否则返回 0
extern unsigned char tolower (unsigned char)	将大写字符转换为小写形式, 如果字符不在 “A” 到 “Z” 之间, 则不作变换而直接返回这个字符
extern unsigned char toupper (unsigned char)	将小写字符转换为大写形式, 如果字符不在 “a” 到 “z” 之间, 则不作变换而直接返回这个字符
extern unsigned char toint (unsigned char)	将 ASCII 字符的 0~9, A~F (大小写无关) 转换为十六进制数字, 返回值 0H~9H 由 ASCII 字符的 0~9 得到, 返回值 0AH~0FH 由 ASCII 字符 A~F (大小写无关) 得到

(4) 标准函数 STDLIB.H

标准函数 STDLIB.H 主要包含一些字符串转换为数值类型、开辟内存空间、产生随机数的函数，详见表 2.8。

表 2.8 标准函数 STDLIB.H

函 数	功 能
float atof (void *string)	将字符串转换成浮点数
int atoi (void *string)	将字符串转换成整数
unsigned long strtod (const char *string,char **ptr)	将字符串转换成整数
long strtol (const char *string,char **ptr,unsigned char base)	将字符串转换成成长整数
unsigned long strtoul (const char *string,char **ptr)	将字符串转换成无符号整数
void *calloc (unsigned int num,unsigned int len)	申请有 num 个元素的，每个元素的大小为 len 的数组
void *malloc (unsigned int size)	申请一块大小为 size 的内存
void *realloc (void xdata *p,unsigned int size)	重新申请一块大小为 size 的内存
void free (void xdata *p)	释放一块被申请的内存
void init_mempool (void xdata *p,unsigned int size)	初始化内存池
int rand (void)	取一个 0~32767 之间的随机数
void srand (int seed)	根据种子 seed 的值取随机数

(5) 字符串函数 STRING.H

字符串相关的库函数如表 2.9 所示。

表 2.9 字符串 STRING.H

函 数	功 能
void *memcpy (void *dest,void *src,char c, int len)	该函数将从 src 开始的字节复制到 dest 所指向的位置。该函数可以复制多个字节。调用该函数以后，字符串会被复制，直到 c 字符被复制，或者复制了 len 个字节为止。该函数返回值为指向最后被复制到 dest 的字节之后的那个字节的指针
void *memchr (void *buf,char c, int len)	该函数将从 buf 开始，长度为 len 的字符串中查找字符 c。如果查找到字符 c，那么返回值为指向该字符的指针，如果未找到，返回值为 NULL。或者为最后被复制到 dest 的字节之后的那个字节的指针
char memcmp (void *buf1,void *buf2,int len)	该函数将比较 buf1 和 buf2 开始，长度为 len 的 2 个字符串的大小。如果 buf1 指向的字符串小于 buf2 指向的字符串，那么返回一个小于 0 的值；如果大于，则返回一个大于 0 的值；如果相等，那么返回 0
void memcpy (void *dest,void *src,int len)	该函数将把长度为 len，起始地址为 src 的字符串复制到起始地址为 dest 的地方，返回值为 dest
void *memmove (void *dest,void *src,int len)	该函数把长度为 len，起始地址为 src 的字符串复制到起始地址为 dest 的地方，如果存储区域重叠，该函数保证 src 指向的内容能被复制到 dest。返回值为 dest
void *memset (void *buf,char c,int len)	该函数把长度为 len，起始地址为 buf 的所有字符的内容写为 c
void *strcat (char *dest ,char *src)	该函数将 src 指向的字符串复制到 dest 指向的字符串之后。返回值为 dest
char *strncmp (char *string1,char *string2,int len)	该函数比较字符串 string1 和 string2 的前 len 个字符的大小
char *strncpy (char *dest,char *src,int len)	该函数将 src 字符串中最多 len 字节长度的字符复制到 dest 中
char *strpbrk (char *string,char *set)	该函数在字符串 src 中查找是否有字符数组 set 中的任何一个元素。返回值为查找到的字符的指针或者 NULL（未查找到的时候）

(续表)

函 数	功 能
int strops (const char*string,char c)	该函数在字符串 string 中查找字符 c, 返回查找到的字符的序号。 如果没有查找到, 则返回-1
char *strrch (const char*string,char c)	功能与 strchr 相同, 只不过是反向查找
char *strrprk (char *string,char *set)	功能与 strprk 基本相同, 只不过是反向查找
char *strpos (const char *string,char *c)	功能与 strops 基本相同, 只不过是反向查找

(6) C51 其他库函数介绍

C51 中还有其他库函数供开发者调用, 如绝对地址访问、内部函数、变量参数表和全程跳转表等库函数。

① 绝对地址访问 ABSACC.H

```
#define CBYTE ((unsigned char volatile code*) 0)
#define DBYTE ((unsigned char volatile idata*) 0)
#define PBYTE ((unsigned char volatile pdata*) 0)
#define XBYTE ((unsigned char volatile xdata*) 0)
```

功能: 上述宏定义用来对 8051 系列单片机的存储器空间进行绝对地址访问, 可以作为字节寻址。CBYTE 寻址 CODE 区, DBYTE 寻址 DATA 区, PBYTE 寻址分页 XDATA 区, XBYTE 寻址 XDATA 区。

```
#define CWORD ((unsigned char volatile code*) 0)
#define DWORD ((unsigned char volatile idata*) 0)
#define PWORD ((unsigned char volatile pdata*) 0)
#define XWORD ((unsigned char volatile xdata*) 0)
```

功能: 这个宏与前面的一些宏类似, 只不过数据类型为 unsigned int 类型。

② 内部函数 INTRINGS.H

```
unsigned char _crol_ (unsigned char c,unsigned char b)
unsigned char _irol_ (unsigned int i,unsigned char b)
unsigned char _irol_ (unsigned long i,unsigned char b)
```

将第 1 个参数循环左移 n 位。

```
unsigned char _cror_ (unsigned char c,unsigned char b)
unsigned char _iror_ (unsigned int i,unsigned char b)
unsigned char _iror_ (unsigned long i,unsigned char b)
```

将第一个参数循环右移 n 位。

```
void _nop_ (void)
```

产生一个单片机的空操作指令, 一般用于延时或者等待。

```
bit _testbit_ (bit x)
```

产生一个 8051 单片机的位操作指令 JBC。该函数对字节中的一个比特进行测试, 如果该比特为 1 则返回 1, 同时将该比特复位为 0; 否则直接返回 0。

③ 变量参数表 STDARG.H

C51 编译器允许再入的函数的参数个数和类型是不变的, 可以用简略形式 (记号

“...”），这时参数表的长度和参数的数据类型在定义时是未知的。头文件 `stdarg.h` 中定义了处理函数参数表的宏，利用这些宏，程序可以识别和处理变化的参数。

```
typedef char *va_list
```

`va_list` 被定义成指向参数表的指针。

```
type va_arg (argptr,type)
```

`va_arg` 从 `argptr` 指向的参数表返回类型为 `t` 的当前参数。

```
void va_start (argptr,prevparm)
```

`va_start` 初始化指向参数的指针。

```
void va_end (argptr)
```

`va_end` 关闭参数表，结束对可变参数表的访问。

④ 全程跳转 SETJMP.H

该头文件的函数可以用于正常的系列函数调用和函数结束，它允许从深层函数调用中直接返回。

```
int setjmp (jmp_buf env)
```

该函数将程序执行的当前环境状态信息存入变量 `env` 中，以便嵌套调用的底层函数使用 `longjmp` 将执行控制权直接返回到调用 `setjmp` 语句的下一条语句。当直接调用 `setjmp` 时返回值为 0；当从 `longjmp` 调用时，返回非 0 值。函数 `setjmp` 只能在 `if` 或者 `switch` 语句中调用一次。

```
void longjmp (jmp_buf env,int retval)
```

`longjmp` 恢复调用 `setjmp` 时存在 `env` 中的状态。程序从调用 `setjmp` 语句的下一条语句执行。参数 `val` 为调用 `setjmp` 的返回值。在调用 `longjmp` 后，由 `setjmp` 调用的函数中的所有自动变量的值都将被改变。

2.2 学习实例

实例一 用 P1 口、P2 口分别显示二进制加、减法结果

1. 实例说明

设两个无符号二进制数 01100101B (65H) 和 00111000B (38H)，将两个数相加结果直接送寄存器 P1，将两个数相减结果直接送寄存器 P2。P1 口和 P2 口分别接 8 只发光二极管，点亮者为 0，不亮者为 1，观察结果。

2. 仿真电路

用 P1 口、P2 口分别显示二进制加、减法结果仿真电路如图 2.10 所示。

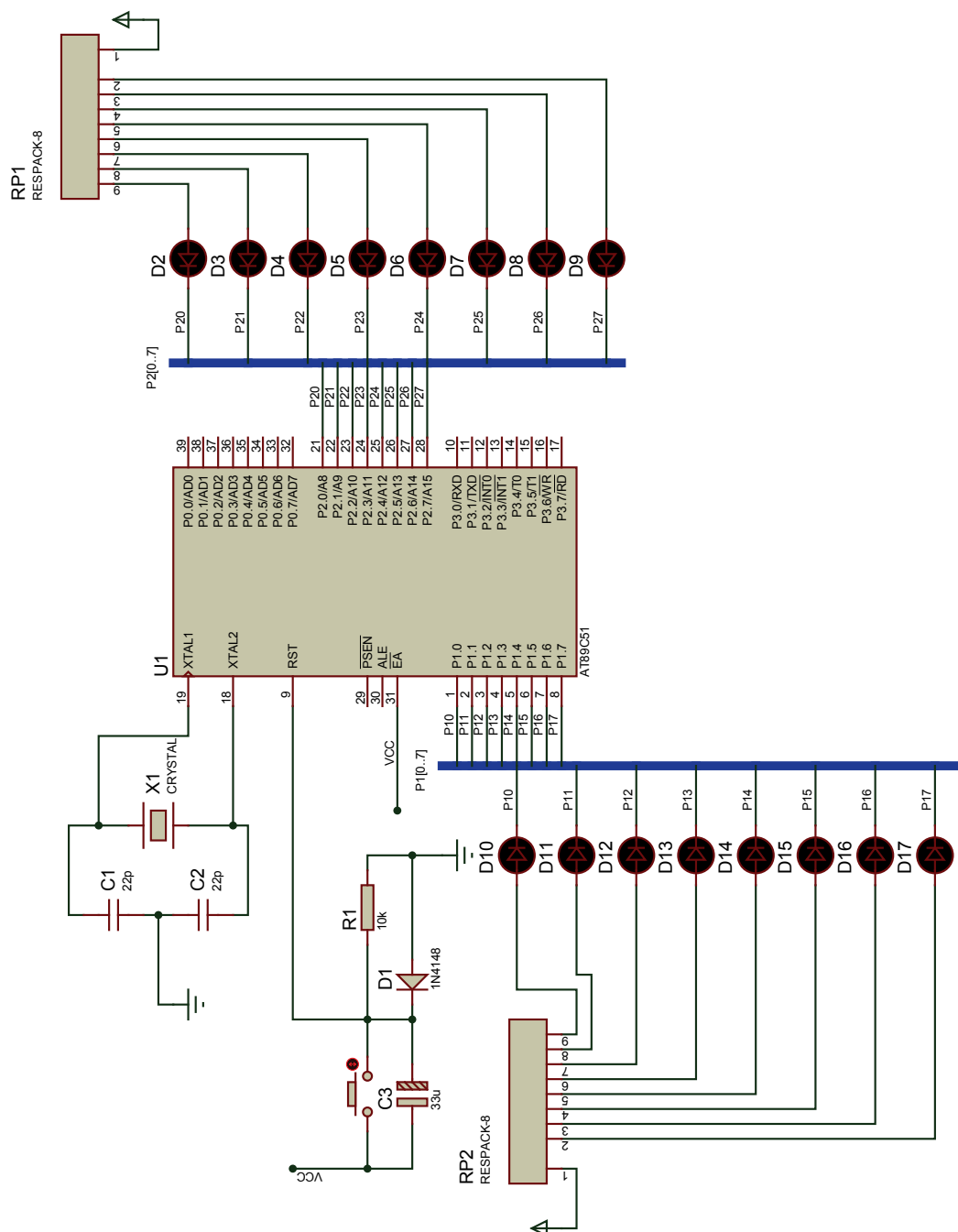


图 2.10 用 P1 口、P2 口分别显示二进制数加、减法结果仿真电路

3. 程序设计

汇编程序如下：

```

    ORG 0000H
    LJMP MAIN
    ORG 0040H
MAIN:CLR C    ;清进位标志
    MOV A,#01100101B ;加数送累加器 A
    ADD A,#00111000B ;两数相加
    MOV P1,A      ;相加结果 10011101 送 P1 口
    CLR C        ;清借位标志
    MOV A,#01100101B ;被减数送累加器 A
    SUBB A,#00111000B ;两数相减
    MOV P2,A      ;相减结果 00101101 送 P2 口
    SJMP $        ;无限循环
    END          ;结束汇编

```

C 程序如下：

```

#include<at89x51.h>
void main()
{
    unsigned char x,y;//定义无符号变量
    x=0x65;//x 赋值 65H (01100101B)
    y=0x38;//x 赋值 38H (00111000B)
    P1=x+y;//P1=10011101B, P1.6、P1.5、P1.1 的 LED 被点亮
    P2=x-y;//P2=00101101B, P2.7、P2.6、P2.4、P2.1 的 LED 被点亮
    while(1); //无限循环
}

```

实例二 用 P2 口实现左右跑马灯效果

1. 实例说明

要求实现 LED 左右流水灯效果，P2 口接八个灯作跑马灯。采用了寄存器中的中间数。

2. 仿真电路

用 P2 口实现左右跑马灯效果仿真电路如图 2.11 所示。

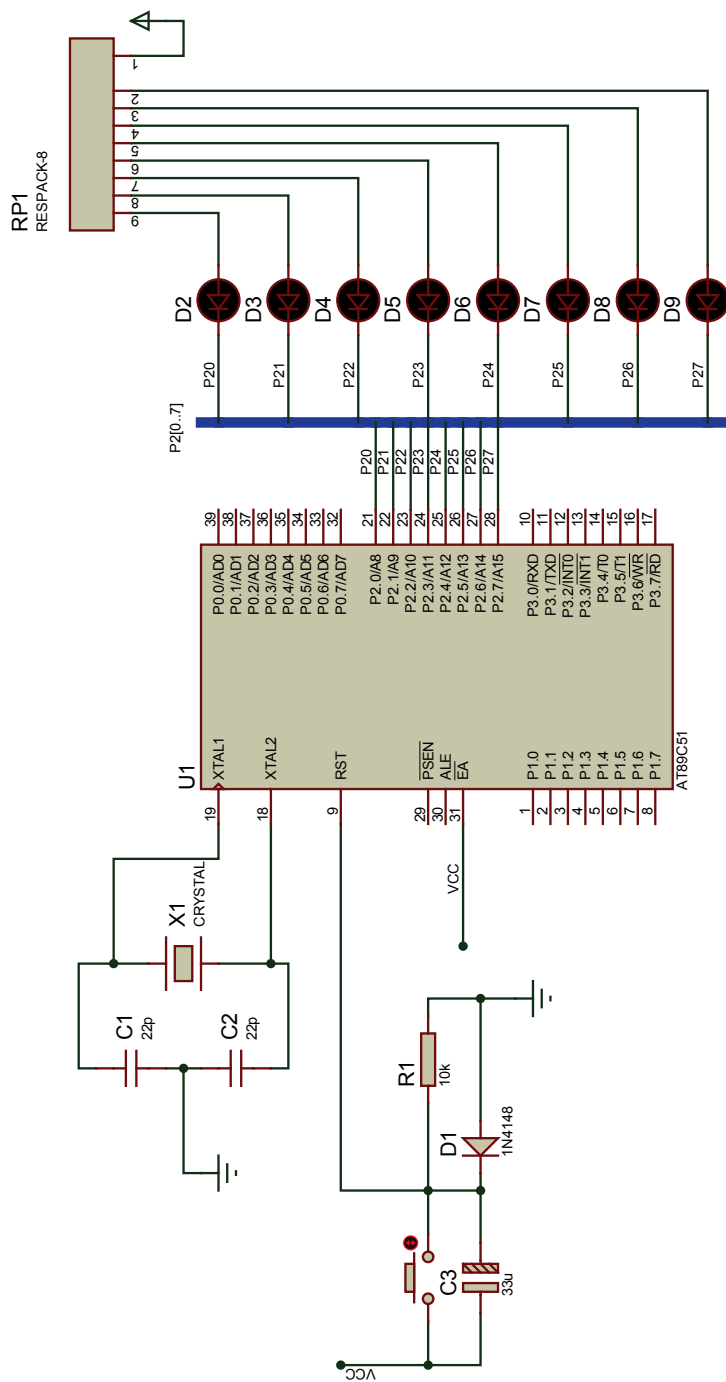


图 2.11 用 P2 口实现左右跑马灯效果仿真电路

3. 程序设计

汇编程序如下：

```

ORG    0000H
AJMP   START
ORG    0040H
START:
    MOV A,#0FFH
    CLR C
    MOV R2,#08H    ;循环八次
LOOP:  RLC A        ;带进位左移
    MOV P2,A        ;输出到 P2 口
    CALL DELAY      ;延时一段时间
    DJNZ R2,LOOP    ;反复循环
    MOV R2,#07H    ;再往回循环
LOOP1: RRC A        ;带进位右移
    MOV P2,A        ;输出到 P2 口
    CALL DELAY      ;延时一段时间
    DJNZ R2,LOOP1   ;反复循环
    JMP START       ;重新开始
DELAY:  MOV R3,#20   ;延时子程序
D1:     MOV R4,#20
D2:     MOV R5,#248
        DJNZ R5,$
        DJNZ R4,D2
        DJNZ R3,D1
        RET
END

```

C 程序如下：

```

#include <reg52.h>
char LED;
/*****延时函数*****/
void delay(unsigned int i)
{
    unsigned char j;
    for(i;i>0;i--)
        for(j=255;j>0;j--);
}
main()
{
    unsigned char k;
    while (1)
    {
        LED=0xfe;
        for (k=0;k<8;k++)
        {
            P2=LED;

```

```

delay(500);
LED=LED<<1;    // 左移
LED=LED|0x01;   // 移位后, 后面的位为高电平
if(LED==0x7f) break;    //退出 FOR 循环
}
for(k=0;k<8;k++)
{
    P2=LED;
    delay(500);
    LED=LED>>1;    // 右移
    LED=LED|0x80;   // 移位后, 后面的位为高电平
}
}
}

```

实例三 用查表法实现 P2 口接的 8 只 LED 灯花样显示

1. 实例说明

通过查表法获得 1 个字节的值然后赋值给 P2 端口, 实现 LED 灯花样显示。

2. 仿真电路

用查表法实现 P2 口接的 8 只 LED 灯花样显示仿真电路图 2.12 所示。

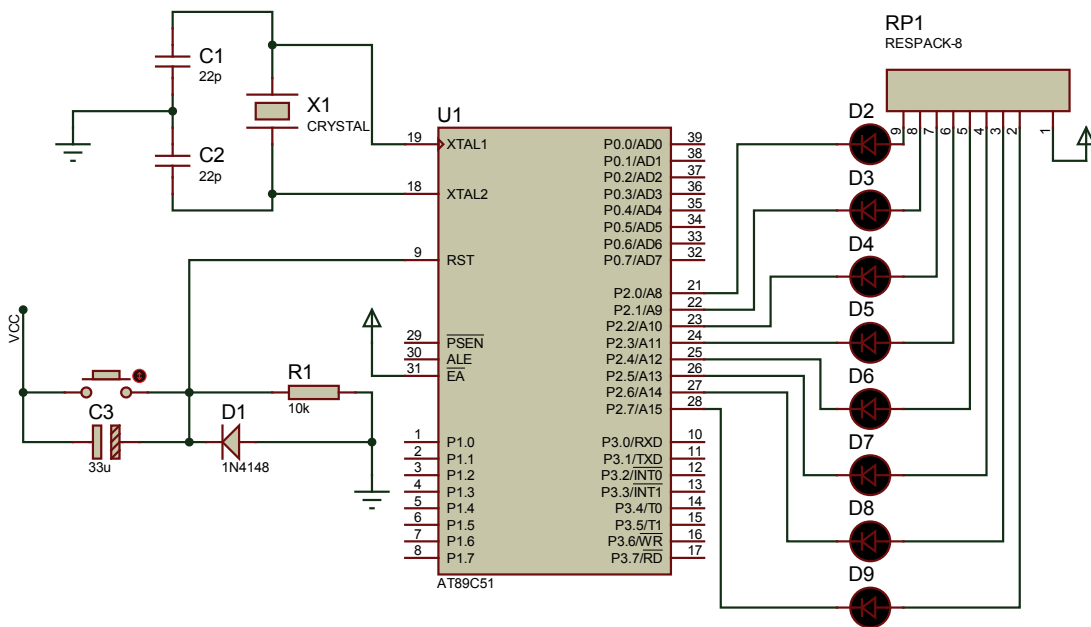


图 2.12 用查表法实现 P2 口接的 8 只 LED 灯花样显示仿真电路

3. 程序设计

汇编程序如下：

```

ORG    0000H
START:MOV    DPTR,#TABLE    ;将表的首地址存入数据指针
LOOP:CLR     A
        MOVC  A,@A+DPTR     ;查表
        CJNE  A,#01,LOOP1   ;取出的码是否 01H? 否则跳到 LOOP1
        JMP   START
LOOP1:MOV     P2,A           ;取出的值输出到 P2 端口
        MOV   R3,#20        ;用于改变延时长度
        CALL  DELAY
        INC   DPTR
        JMP   LOOP
DELAY: MOV    R4,#20
D1:      MOV   R5,#248
        DJNZ  R5,$
        DJNZ  R4,D1
        DJNZ  R3,DELAY
        RET
TABLE: DB  7fH,0bfH,0dfH,0efH,0f7H,0fbH
        DB  0fdH,0feH,0ffH,0ffH,00H,00H
        DB  55H,55H,0aaH,0aaH
        DB  01H;结束码
        END

```

C 程序如下：

```

#include<reg52.h> //包含头文件
unsigned char code table[]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb,0xfd, 0xfe, 0xff,
0xff, 0x00, 0x00, 0x00, 0x55, 0xaa,0xaa};
//rom 允许情况可以无限添加

void Delay(unsigned int t); //函数声明
void main (void)
{
    unsigned char i; //定义一个无符号字符型局部变量 i 取值范围 0~255
    while (1)        //主循环
    {
        for(i=0;i<16;i++) //加入 for 循环,表明 for 循环大括号中的程序循环
                           //执行 16 次,表明表格中有 16 个元素
        {
            P2=table[i];
            Delay(30000);
        }
    }
}

```



```
void Delay(unsigned int t)
{
    while(--t);
}
```

本章小结

本章介绍了 MCS-51 单片机的寻址方式、指令系统以及用汇编和 C51 语言程序设计方法。并对汇编语言程序设计基础、C51 语言的数据类型和存储类型与流程控制语句、C51 函数以及 C51 程序设计的其他问题进行了说明。

习题二

一、填空题

1. 在 R7 初值为 00H 的情况下, DJNZ R7, rel 指令将循环执行_____次。
2. MCS-51 的两条查表指令是_____和_____。
3. 指令 MOV C, 20H 的源操作寻址方式为_____寻址。
4. JZ e 是双字节指令, 其操作码地址为 1000H, e=20H, 它的转移目的地址为_____。
5. 如果(A)=58H, (R1)=49H, (49H)=79H, 执行指令 XCH A, @R1 之后; 结果(A)=_____, (49H)=_____。
6. 在变址寻址方式中, 以_____作变址寄存器, 以_____或_____作基址寄存器。
7. LJMP 的跳转范围是_____, AJMP 的跳转范围是_____, SJMP 的跳转范围是_____。
8. 若 A 中的内容为 68H, 那么 P 标志位为_____。

二、单项选择题

1. MOVX A,@DPTR 指令中源操作数的寻址方式是 ()。
A. 寄存器寻址
B. 寄存器间接寻址
C. 直接寻址
D. 立即寻址
2. ORG 0003H
LJMP 2000H
ORG 000BH
LJMP 3000H
当 CPU 响应外部中断 0 后, PC 的值是上述中的哪一个 ()。
A. 0003H
B. 2000H
C. 000BH
D. 3000H


```
ANL A, #17H
ORL 17H, A
XRL A, @R0
CPL A
```

2. 下列程序段经汇编后, 从 1000H~100AH 存储单元的内容是什么? 把答案填到下面的表里。

```
ORG 1000H
TAB1 EQU 1234H
TAB2 EQU 3000H
DB "START"
DW TAB1,TAB2,70H'
```

1000H	
1001H	
1002H	
1003H	
1004H	
1005H	
1006H	
1007H	
1008H	
1009H	
100AH	

3. 阅读下面程序, 说明其功能。

```
MOV A, R1
MOV B, R2
CJNE A, B, BJ1
BJ1: JC BJ2
MOV A, R2
BJ2: MOV P1, A
```

功能: _____。

4. 阅读下面程序, 说明其功能。


```
MOV DPTR,#1000H
MOV R0,#30H
LOOP:MOVX A,@DPTR
MOV @R0,A
```

```
INC    DPTR
INC    R0
CJNE   R0, #71H, LOOP
RET
```

功能：_____

_____。

四、编程题

1. 编写程序，查找在片内 RAM 中的 20H~50H 单元中出现 00H 的次数，并将查找结果存入 51H 单元。
2. 编写程序，将外部数据存储器中的 5000H~50FFH 单元全部清零。
3. 已知在累加器 A 中存放一个 BCD 数 (0~9)，请编程实现一个查平方表的子程序。
4. 将字节地址 30H~3FH 单元的内容逐一取出减 1，然后再放回原处，如果取出的内容为 00H，则不要减 1，仍将 0 放回原处。 

第3章 单片机中断系统、定时器/计数器及串行口

学习目标

掌握 51 单片机中断系统的结构及中断处理过程，学会编写利用中断方式实现数据输入/输出程序，了解外部中断源扩展方法。掌握定时器/计数器结构及四种工作方式；学会使用定时器/计数器的定时或计数功能，并编写相应控制程序；了解串行通信的基本概念，掌握串行口的基本结构及工作方式。

重点难点

51 单片机中断系统结构及中断处理过程，编写中断处理程序。定时器/计数器的结构及四种工作方式，编写相应的控制程序。串行口的结构，串行口四种工作方式的工作过程及发送和接收程序的编写。

3.1 知识结构

3.1.1 中断系统

1. 中断的概念

中断：CPU 正在执行程序的过程中，由于某种随机事件的发生，有必要暂停该程序的执行，转而去执行相应的处理程序，待处理程序结束之后，再返回原程序断点处继续运行的过程。

中断系统：实现中断处理功能的部件。

中断源：提出中断申请的来源。中断源一般有外设、定时时钟、故障源等。

主程序与中断服务程序：CPU 执行的当前程序称为主程序。CPU 转去对随机事件的处理程序，称为中断服务程序。

中断优先级：当多个中断源同时申请中断时，为了使 CPU 能够按照用户的规定先处理最紧急的，然后再处理其他事件，中断系统设置有中断优先权排队电路，通过用户的设置，排在前面的中断源称为高级中断，排在后面的称为低级中断。

中断嵌套：当 CPU 响应某一中断源请求而进入中断处理时，若更高级别的中断源发出申请，则 CPU 暂停现行的中断服务程序，去响应优先级更高的中断，待更高级别的中断处理完毕后，再返回低级中断服务程序，继续原先的处理，这个过程称为中断嵌套。低级中断不能中断优先级高的中断，同级中断不能中断优先级相同的中断。



2. MCS-51 中断系统结构

MCS-51 单片机有 5 个中断源，其中两个外部中断源 $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 、两个定时器/计数器 (T0、T1) 和一个串行口中断。这五个中断源都可以控制为允许或禁止状态，通过对 IP 寄存器的设置可以设为高、低两个优先级，可形成中断嵌套。MCS-51 单片机的中断系统由与中断有关的特殊功能寄存器、中断入口、顺序查询逻辑电路等组成。MCS-51 中断系统结构图，如图 3.1 所示。

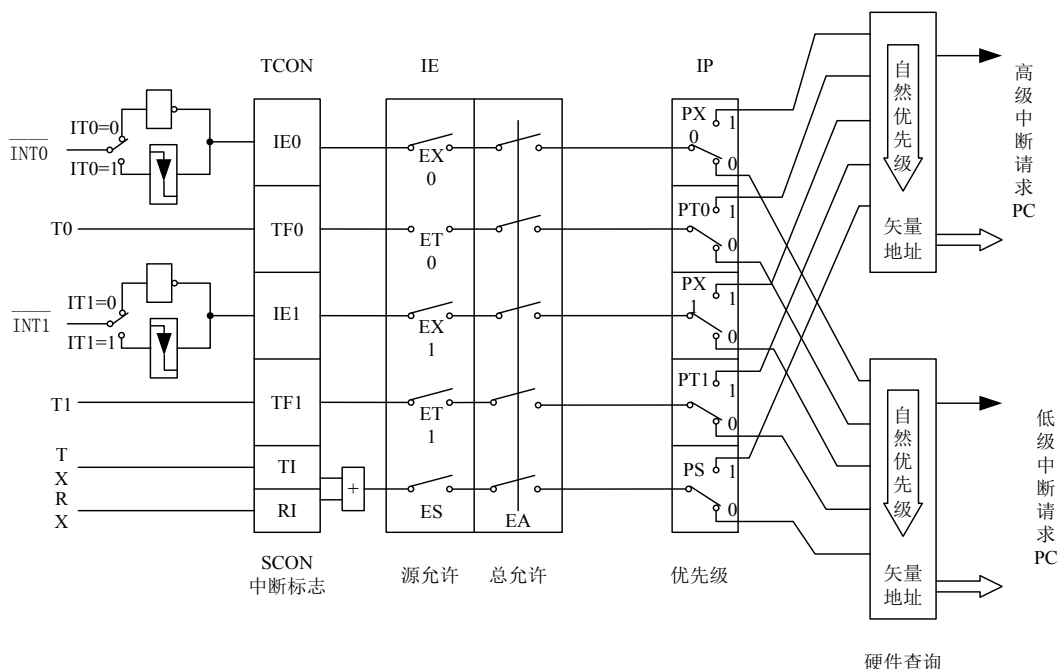


图 3.1 MSC-51 中断系统结构图

MSC-51 的中断系统有 4 个特殊寄存器：定时器控制寄存器 TCON、串行口控制寄存器 SCON、中断允许寄存器 IE、中断优先级寄存器 IP。

其中，TCON 和 SCON 只有一部分位用于中断控制。通过对以上各特殊功能寄存器的有关位进行置位（置 1）或复位（清 0）等操作，实现各种中断控制功能。

（1）定时器控制寄存器 TCON（地址 88H）

TCON 为中断请求标志及外部中断触发方式选择寄存器，其格式如下：

位	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TCON	TF1		TF0		IE1	IT1	IE0	IT0

TF0、TF1、IE0、IE1：分别为定时/计数器 T0、定时/计数器 T1、外部中断 0、外部中断 1 的中断请求标志，若有中断请求，相应中断标志位置 1；若无中断请求，该中断标志位清 0。当系统复位后，TCON 各位均清 0。

IT0、IT1：外部中断 0 和外部中断 1 的触发方式控制位。若采用边沿（下降沿）触发，

该位置 1；若采用低电平触发，该位清 0。在边沿触发方式中，为了保证 CPU 在两个机器周期内检测到先高后低的负跳变，输入高低电平的持续时间至少要保持一个机器周期。

(2) 串行口控制寄存器 SCON 中的中断标志位（地址为 98H）

SCON 为串行口控制寄存器，SCON 的低 2 位 TI 和 RI 为串行口的接收中断和发送中断标志。

TI：串行口发送中断标志位。发送完一帧串行数据后，由硬件置 1，向 CPU 提出中断请求。但 CPU 响应中断时不会对 TI 清 0，必须由软件（指令）清 0。

RI：串行口接收中断标志位。接收完一帧串行数据后，硬件置 1。同时串行口接收器向 CPU 提出中断请求。同样，CPU 响应中断是不会对 RI 清 0，必须由软件（指令）清 0。

(3) 中断允许寄存器 IE（地址 A8H）

每个中断源中断的允许与禁止由中断允许寄存器 IE 中的某一位控制，其格式如下：

位	AFH		ACH		ABH	AAH	A9H	A8H
IE	EA			ES	ET1	EX1	ET0	EX0

EA：CPU 中断允许标志。EA=1，CPU 开中断；EA=0，CPU 禁止（屏蔽）所有中断请求。

ES：串行口的中断允许位。ES=1，允许串行口中断；ES=0，禁止串行口中断。

ET1：定时/计数器 T1 的中断允许位。ET1=1，则允许 T1 中断；ET1=0，禁止 T1 中断。

EX1： $\overline{INT1}$ 的中断允许位。EX1=1，允许外部中断 1 中断；EX1=0，禁止 $\overline{INT1}$ 中断。

ET0：定时/计数器 T0 的中断允许位。ET0=1，则允许 T0 中断；ET0=0，禁止 T0 中断。

EX0： $\overline{INT0}$ 的中断允许位，EX0=1，允许外部中断 0 中断；EX0=0 则禁止 $\overline{INT0}$ 中断。

MCS-51 单片机系统复位后，IE 中各位均被清 0，即禁止所有中断。

(4) 中断优先级控制寄存器 IP（地址 B8H）

MCS-51 单片机有两个中断优先级，由 IP 寄存器管理，可实现二级中断嵌套。一个中断源对应 IP 中的一位。其格式如下：

位	BCH			ABH	AAH	A9H	A8H	
IP				PS	PT1	PX1	PT0	PX0

PS、PT1、PX1、PT0、PX0：分别为串行口中断、定时器/计数器 T1、外部中断 1、定时/计数器 T0、外部中断 0 的优先级控制位。对应位置 1，该中断源为高优先级；对应位清 0，该中断源为低优先级。

MCS-51 单片机系统复位后，IP 中各位均被清 0，各中断源均设为低优先级中断。

若同时收到几个同一优先级的中断请求时，由内部的查询确定优先顺序，优先响应先查询的中断请求。查询顺序如下：

外部中断 0→T0 溢出中断→外部中断 1→T1 溢出中断→串行口中断

3. 中断的处理过程

中断处理过程分为 4 个阶段：中断请求、中断响应、中断服务和中断返回。

MCS-51中断的处理过程如图3.2所示。

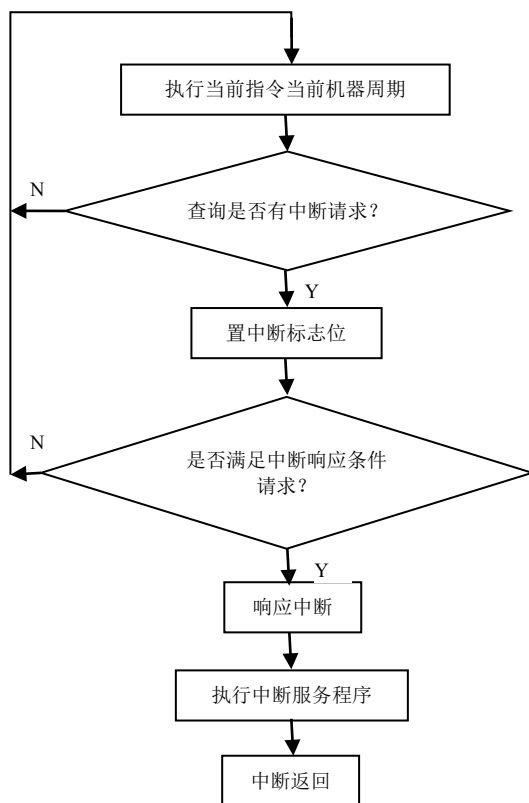


图 3.2 中断的处理过程

(1) 中断请求

CPU在每个机器周期结束时查询中断源是否有中断申请，如果有中断请求，则自动设置相应中断请求标志位；如果没有，则继续当前任务。

(2) 响应中断

设置相应的优先级状态触发器，保护现行程序断点地址，将断点PC压入堆栈，进入指定的中断服务程序入口地址，转到中断服务程序运行。

(3) 执行中断服务程序

在中断服务程序中不仅要完成相应的服务任务，而且要考虑现场保护与现场恢复，以便保护主程序中不应破坏的数据。

(4) 中断返回

通过执行一条中断返回指令RETI完成，该指令清除响应时设置的优先级状态触发器、恢复主程序断点地址，即把堆栈的内容送给PC，继续执行主程序。

CPU响应某中断请求后，在返回之前必须撤除中断请求。

MCS-51单片机的中断系统在响应中断后，能够自动清除两个定时器的中断请求标志TF1、TF0和边沿触发下的两个外部中断请求标志IE1、IE0，对电平触发下的两个外部中断请求标志IE1、IE0，必须撤除引脚上的请求信号，才能根本上对请求标志清0，RI和TI由软

件清0。

4. 汇编语言中断处理程序结构

以包含 T0 中断服务子程序的程序结构为例：

```

        ORG    0000H
        LJMP   MAIN
        ORG    000BH; T0 中断服务程序入口
        LJMP   INTT0
        ORG    0100H
MAIN:   .SETB EA; 主程序
        SETB T0
        .
        .
INTT0:  PUSH    ACC; 保护现场
        PUSH    DPH
        PUSH    DPL
        PUSH    PSW
        中断源服务
        POP     PSW; 恢复现场
        POP     DPL
        POP     DPH
        POP     ACC
        RETI
END

```

5. C51 中断处理程序结构

```

#include<re51.h>
    .
    .
void main()
{
    EA=1;
    ET0=1;
    .
    .
    while(1)
    {
        .
        .
    }
}

void intt0(void) interrupt 1 using 2
{
    .
    .
}

```


TMOD用于控制定时器/计数器的启动方式、定时器/计数器的选择、工作方式的选择。此寄存器只能字节寻址，复位时，TMOD=00H，其高4位控制T1，低4位控制T0，其格式如下：

位 TMOD	T1 控制字段				T0 控制字段			
	GATE	C/\bar{T}	M1	M0	GATE	C/\bar{T}	M1	M0

GATE：门控位。

M1 M0：工作方式选择位。

共有 4 种编码，对应于 4 种工作方式的选择，如表 3.2 所示。

C/\bar{T} ：定时、计数选择。

表 3.2 由 M1 和 M0 设定定时器/计数器

0	0	方式 0（13 位定时器/计数器）
0	1	方式 1（16 位定时器/计数器）
1	0	方式 2（8 位自动装入计数初值方式）
1	1	方式 3（仅用于 T0，分成两个 8 位定时器/计数器）

（2）定时器控制寄存器 TCON（地址 88H）

此寄存器复位时，TCON=00H。其格式如下：

位	8F	8E	8D	8C	8B	8A	89	88
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TCON 寄存器低 4 位与外部中断有关，高 4 位的功能如下：

TF1、TF0：溢出中断请求标志。TF1 对应 T1，TF0 对应 T0。该位为 1 时，计数器计数溢出，有中断请求；该位为 0 时，无计数溢出中断请求。

TR1、TR0：计数运行控制位。TR1 对应 T1，TR0 对应 T0。该位为 1 时，启动定时器/计数器；该位为 0 时，停止定时器/计数器。

3. 定时器/计数器有四种工作方式

（1）方式 0

当 TMOD 寄存器中 M1M0=00 时，定时器/计数器被设置为工作方式 0。如图 3.4 所示是定时器方式 0 的逻辑电路结构。

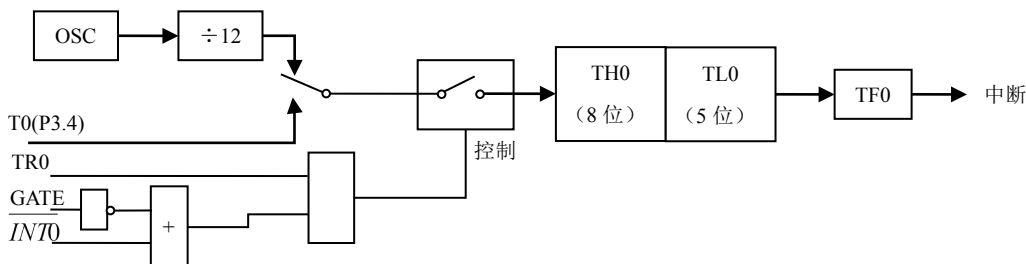


图 3.4 T0 工作方式 0 的电路结构

方式0为13位的计数器，由 $TLx(x=0,1)$ 的低5位和 $THx(x=0,1)$ （高8位）所构成，最大计数值为 2^{13} 。

$TLx(x=0,1)$ 的低5位进位时， $THx(x=0,1)$ 加1， $THx(x=0,1)$ 最高位进位（即溢出）时，设置 $TFx(x=0,1)=1$ ，申请中断。若CPU响应中断，系统自动对 $TFx(x=0,1)$ 复位。

$C/\overline{T}=0$ ，为定时功能； $C/\overline{T}=1$ ，为计数功能，对外部脉冲输入端T0和T1输入的脉冲计数。

定时器T0的启动控制由门控位GATE、启动位TR0、引脚 $\overline{INT0}$ 的逻辑组合确定，GATE=0时，TR0=1时启动计数；GATE=1时，TR0=1和 $\overline{INT0}=1$ 时启动计数。

（2）方式1

当TMOD寄存器中M1M0=01时，定时器/计数器被设置为工作方式1。

方式1与方式0基本相同，差别仅仅在于计数器的位数是16位的，由 $TLx(x=0,1)$ （低8位）和 $THx(x=0,1)$ （高8位）所构成，最大计数值为 2^{16} 。

（3）方式2

当TMOD寄存器中M1M0=10时，定时器/计数器被设置为工作方式2。如图3.5所示是定时器方式2的逻辑结构。

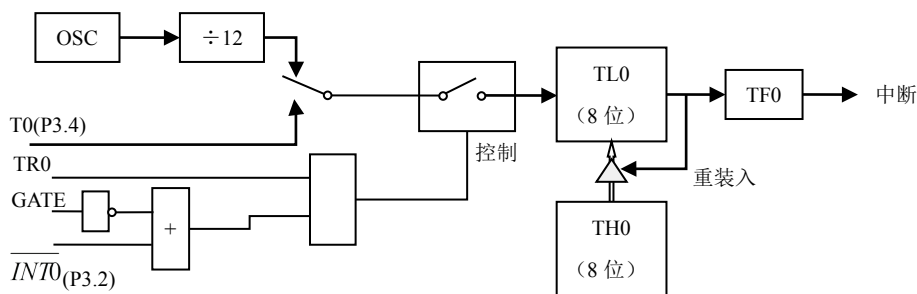


图 3.5 T0（或 T1）工作方式 2 的结构图

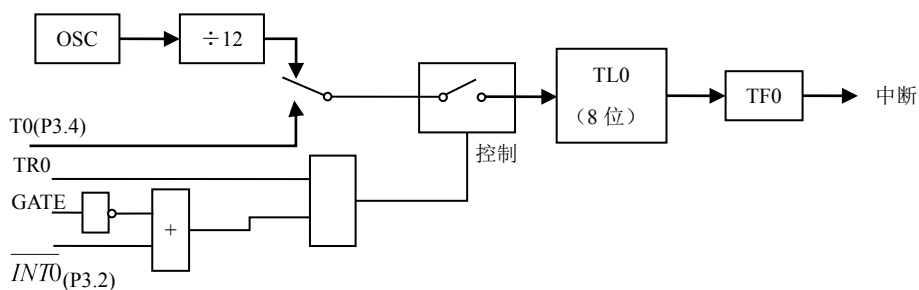
方式2为8位可自动重载的定时器/计数器工作方式， $TLx(x=0,1)$ 作为常数缓冲器，当计数溢出时，除置1溢出中断标志外，还自动将 $THx(x=0,1)$ 的初值送至 TLx 。而在方式1和方式0中，若要进行下一次定时/计数，需要用户重装初值。

（4）方式3

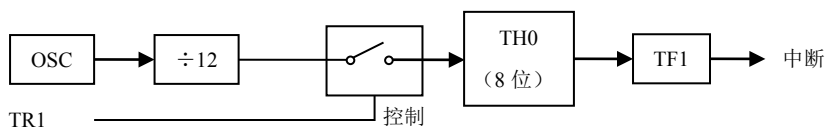
T0设置为方式3时，分成两个独立的8位定时器/计数器TL0和TH0。TL0可以作计数器和定时器使用，T0的各控制位和引脚信号全归它使用，其功能和操作与方式0或方式1完全相同。TH0只能作为定时器使用，它的启动仅由原来T1的启动位TR1控制，当TH0溢出时，置位TF1标志申请中断，中断服务程序入口为001BH。当T0工作在方式3时，T1可以工作在方式0、1、2三种方式。如图3.6所示是定时器工作方式3的逻辑结构图。

4. 定时器/计数器的初始化

定时器/计数器在使用之前均要对其进行初始化。



(a) TL0 作 8 位定时器



(b) TH0 作 8 位定时器

图 3.6 T0 工作方式 3 的逻辑结构图

(1) 初始化的步骤

① 确定是计数模式还是定时模式、工作方式和启动控制方式，并将其写入 TMOD 寄存器。

② 设置定时或计数器的初值：可直接将初值写入 TH0、TL0 或 TH1、TL1 中。16 位计数初值必须分两次写入对应的计数器。

③ 根据要求决定是否采用中断方式：直接对 IE 位赋值。开放中断时，对应位置“1”；采用程序查询方式时，IE 位应清 0 以进行中断屏蔽。

④ 启动定时器工作。若第一步设置为软启动，即 GATE 设置为 0 时，以上指令执行后，定时器即可开始工作。若 GATE 设置为 1 时，还必须由外部中断引脚 $\overline{INT0}$ 和 $\overline{INT1}$ 共同控制，只有当引脚 $\overline{INT0}$ 和 $\overline{INT1}$ 电平为高时，以上指令执行后定时器方可启动工作。定时器一旦启动就按规定的方式定时或计数。

(2) 计数初值的计算

定时或计数方式下计数初值如何确定，定时器选择不同的定时或计数模式，不同的工作方式其计数初值均不相同。

若设最大计数值为 M ，各工作方式下的 M 值为：

方式 0: $M=2^{13}=8192$

方式 1: $M=2^{16}=65536$

方式 2: $M=2^8=256$

方式 3: $M=2^8=256$ ，定时器 T0 分成两个独立的 8 位计数器，所以 TH0、TL0 的 M 均为 256。

MCS-51 的两个定时器均为加 1 计数器，当加到最大值 (00H 或 0000H) 时产生溢出，将 TF 位置 1，可引发溢出中断。因此计数器初值 X 的计算公式为： $X=M-\text{计数值}$ ，式中的 M 由操作模式确定，不同的操作模式计数器的长度不相同，故 M 值也不相同，式中的计数值与定时器的工作方式有关。

计数工作模式时：计数脉冲由外部引入，对外部脉冲进行计数，因此计数值根据要求确定。其计数初值 $X=M$ -计数值。

定时工作模式时：计数脉冲由内部供给，对机器周期进行计数，因此计数脉冲频率为 $f_{\text{count}}=f_{\text{osc}}/12$ 、计数周期 $T=1/f_{\text{count}}=12/f_{\text{osc}}$ ，定时工作方式的计数初值 X ： $X=M$ -计数值= $M-t/T=M-(f_{\text{osc}} \times t)/12$ ，式中为振荡器的振荡频率， t 为要求定时的时间。

例：假设单片机的晶振频率为 $f_{\text{osc}}=6\text{MHz}$ ，产生 1ms 的定时，则方式 0、方式 1 和方式 2 的初值为：

方式 0 初值： $X=2^{13}-(1 \times 10^{-3})/(12/(6 \times 10^6))=7692$

方式 1 初值： $X=2^{16}-(1 \times 10^{-3})/(12/(6 \times 10^6))=65036=\text{FE0CH}$

方式 2 初值： $X=2^8-(1 \times 10^{-3})/(12/(6 \times 10^6))=256-500=-254$

初值为负，说明晶振频率为 $f_{\text{osc}}=6\text{MHz}$ ，不能产生 1ms 的定时。

实例请参考本章实例二、三、四。

3.1.3 串行口

1. 串行通信的基本概念

(1) 并行通信与串行通信

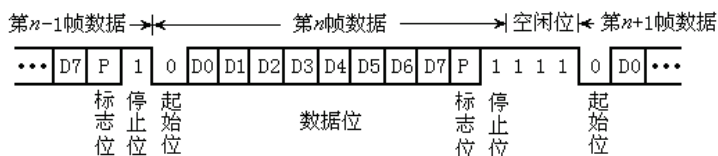
单片机与外界的信息交换称为通信。基本的通信方法有并行通信和串行通信两种。并行通信是指单位信息的各位数据同时传送。它是依靠并行 I/O 口实现的。串行通信是指单位信息的各位按顺序一位一位地传送，它是靠串行接口实现数据传送的。

(2) 同步通信与异步通信

串行通信有两种基本通信方式，即同步通信和异步通信。

同步通信是指发送和接收同步进行，从而实现数据的不间断传送。发送方在数据或字符前面用 1~2 个字节的同步字符指示一帧的开始，同步字符是双方约定的字符，起到提示发送开始和等待的作用。

异步通信是指发送与接收没有使用同步时钟进行同步，一帧数据一般由一位起始位、若干数据位、一位停止位构成，起始位 0 表示字符的开始，最后一个停止位表示字符的结束。数据帧格式如下：



(3) 串行通信的传送方式

串行通信的传送方式有单工传送、全双工传送和半双工传送。

单工是指发送设备和接收设备之间连接的信号线传送方向是单向的。

全双工是指两设备之间连接的信号线传送方向是双向的，并且双向传送能同时进行。

半双工是指两设备之间连接的信号线传送方向是双向的，但两方向的传送不能同时进行。

(4) 波特率的概念

波特率(Baud Rate)是通信中对数据传送速率的规定,指每秒传送数据的位数,单位为波特,即位/秒(b/s)。波特率的倒数称为位传送时间,用 T_d 表示,单位为秒(s)。

2. 串行口的结构

(1) 串行口的总体结构

51 系列单片机的串行接口是一个可编程的全双工串行口,电路结构方框图如图 3.7 所示。它有两个物理上独立的接收、发送缓冲器 SBUF,可同时发送和接收数据,发送缓冲器只能写入不能读出,接收缓冲器只能读出不能写入,两个缓冲器占用同一个地址 99H。还有两个控制寄存器 SCON 和 PCON。

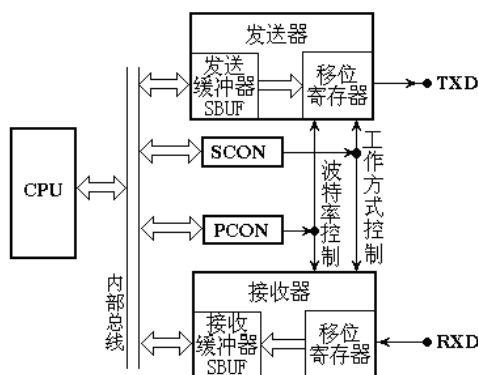


图 3.7 串行接口电路结构方框图

(2) 串行口控制寄存器 SCON (地址 98H)

包含串行口工作方式选择位、接收与发送控制位、串口状态标志位。复位后 SCON=00H。其数据格式如下:

位	9F	9E	9D	9C	9B	9A	99	98
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0、SM1: 串行口工作方式选择位。可设置4种工作方式。

00—方式 0; 01—方式 1; 10—方式 2; 11—方式 3

串行接口的工作方式如表 3.3 所示。

表 3.3 串行接口的工作方式

SM0	SM1	方 式	说 明	波 特 率
0	0	0	移位寄存器	$f_{osc}/12$
0	1	1	10 位异步收发器 (8 位数据)	可变
1	0	2	11 位异步收发器 (9 位数据)	$f_{osc}/64$ 或 $f_{osc}/32$
1		3	11 位异步收发器 (9 位数据)	可变

SM2: 允许方式2、3多机通信控制位, 其功能见表3.4。

REN: 串行接收允许位。由软件设置1允许接收, 设为0禁止接收。

TB8、RB8: 在9位异步通信方式下, 缓冲器只有8位, 故用TB8作为发送的第9位, RB8作为接收的第9位。

TI、RI: 发送中断标志与接收中断标志。当发送完一帧数据后硬件自动置位TI; 当接收完一帧数据后, 若数据满足保留条件, 硬件自动置位RI。若CPU响应中断, 系统不会自动复位TI、RI, 必须由软件清0。

表 3.4 多机控制位功能

串口工作方式	SM2 位	功能说明
方式 0	SM2=0	该位无意义, 设为 0
方式 1	SM2=1	只有接收到有效的停止位, 才将数据送入接收缓冲器保存, 并置 RI=1, 否则数据丢失, 不置位 RI
	SM2=0	无论是否接收到有效的停止位, 都将数据保存, 并置位 RI
方式 2,3	SM2=1	只有接收到第 9 位为 1, 才将数据送入接收缓冲器保存, 并置 RI=1, 否则数据丢失, 不置位 RI
	SM2=0	无论是否接收到第 9 位为 1, 都将数据保存, 并置位 RI

(3) 电源控制寄存器PCON (地址87H)

串行通信只用了其中的最高位, 用来控制串行口的波特率倍增, PCON 寄存器不能进行位寻址, 只能进行字节寻址, 复位后所有位为 0。其格式如下:

位	8E	8D	8C	8B	8A	89	88	87
PCON	SMOD							

SMOD: 波特率倍增位。SMOD=1时, 波特率加倍; SMOD=0时, 波特率不加倍。

3. 串行接口的 4 种工作方式

根据串行通信设计格式和波特率的不同, 51 系列单片机的串行通信方式有 4 种。

(1) 方式 0 (移位寄存器方式)

方式 0 的帧格式如图 3.8 所示。

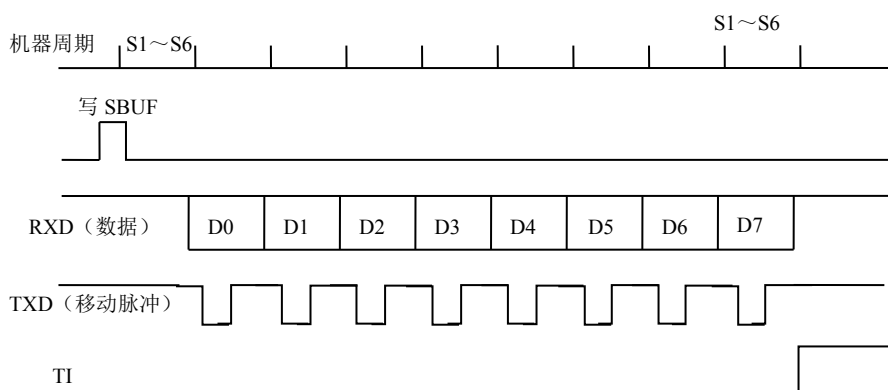
...	D0	D1	D2	D3	D4	D5	D6	D7	...
-----	----	----	----	----	----	----	----	----	-----

图 3.8 方式 0 的帧格式

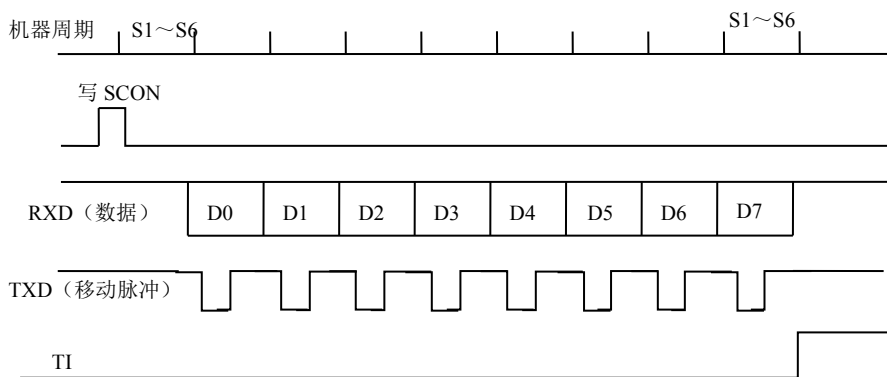
CPU将数据送入发送缓冲器SBUF后, 自动启动串行口发送。8位数据以固定的波特率($f_{osc}/12$), 低位在前高位在后, 从RXD引脚串行输出, TXD引脚发送移位时钟信号(频率为 $f_{osc}/12$)。8位数据发送完毕, 置位TI=1, 申请中断, 通知CPU再发送下一个数据。发送时序如图3.9(a)所示。

软件设置REN=1时, 启动接收过程。串行口以 $f_{osc}/12$ 固定的波特率, 从RXD引脚串行输入数据(低位在前), TXD引脚输出移位时钟信号。当8位数据接收完毕, 将数据送入接收缓冲器SBUF, 并置位RI=1, 申请中断, 通知CPU取走数据。接收时序如图3.9(b)所示。

移位寄存器方式多用于接口的扩展。也可用于短距离的单片机之间的通信。



(a) 方式0发送时序



(b) 方式0接收时序

图 3.9 方式0的发送和接收时序

(2) 方式1 (波特率可变10位异步通信方式)

方式1的帧格式如图3.10所示。

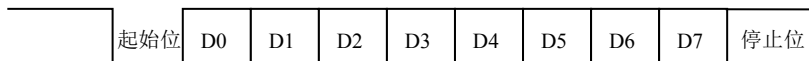
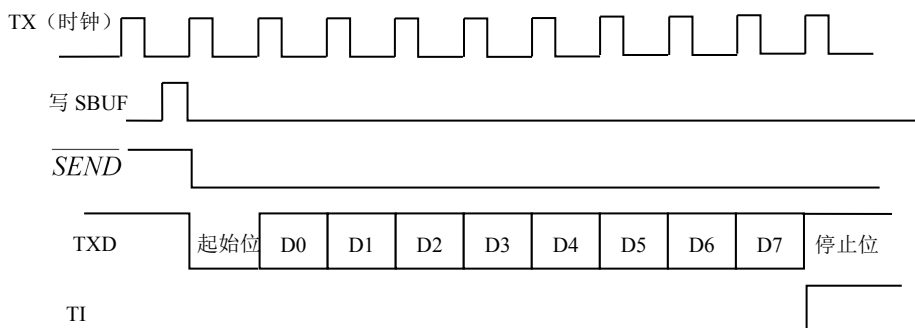


图 3.10 方式1的帧格式

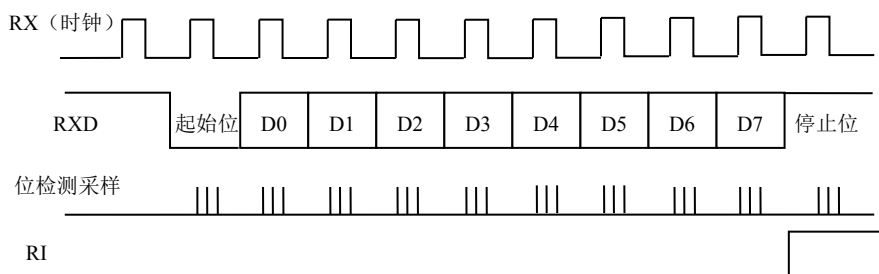
TXD作为串行数据的发送端,以指定的波特率,串行发送一帧10位数据:1个起始位0、8个数据位、1个停止位。发送开始时,内部发送控制信号 $\overline{\text{SEND}}$ 变为有效,将起始位向TXD输出,此后,每经过一个TX时钟周期,便产生一个移位脉冲,并由TXD输出一个数据位。发送数据位全部发送完毕后,置1中断标志位TI,然后 $\overline{\text{SEND}}$ 信号失效。发送时序如图3.11(a)所示。RXD作为数据的接收端,波特率由T1提供,计算公式为:

$$\text{波特率} = \frac{2^{\text{SMOD}}}{32} \times T_1 \text{的溢出率} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{\text{osc}}}{12(256-x)}$$

当检测到起始位的负跳变时，开始接收。接收时定时控制信号有 2 种，一种是接收移位时钟（RX 时钟），它的频率和传送的波特率相同。另一种是位检测器采样脉冲，它的频率是 RX 时钟的 16 倍。在 1 位数据期间，有 16 个采样脉冲，一波特率的 16 倍的速率采样 RXD 引脚状态，当采样到 RXD 端从 1 到 0 的跳变时就启动检测器，接收的值是 3 次连续采样（第 7、8、9 个脉冲采样）取其中 2 次相同的值，以确认是否是真正的起始位（负跳变）的开始，这样能较好地消除干扰引起的影响，以保证可靠无误的开始接收数据。当确认起始位有效时，开始接收 1 帧信息。接收每一位数据时，也都进行 3 次连续采样（第 7、8、9 个脉冲采样），接收的值是 3 次采样中至少 2 次相同的值，以保证接收到的数据位的准确性。接收时序如图 3.11（b）所示。



(a) 方式 1 发送时序



(b) 方式 1 接收时序

图 3.11 方式 1 的发送和接收时序

(3) 方式 2（波特率固定 11 位异步通信方式）

方式 2 的帧格式如图 3.12 所示。

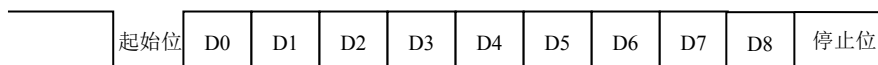


图 3.12 方式 2、方式 3 的帧格式

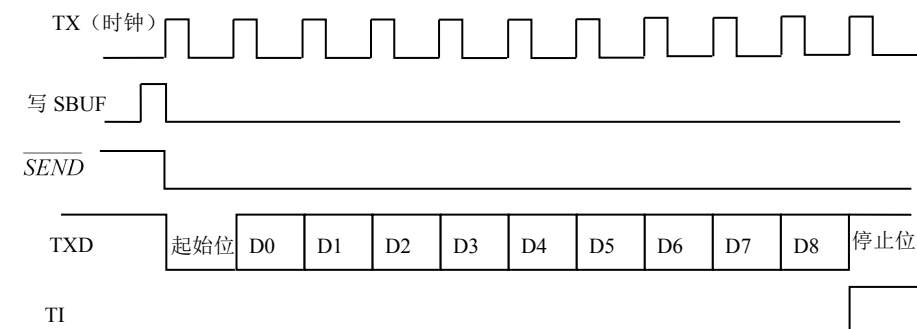
TXD 作为串行数据的发送端，以固定的波特率，串行发送一帧 11 位数据：1 个起始位 0、

8位数据位、TB8提供的第9位、1个停止位1。第9位可作为各种意义的标志，如作为奇偶校验位，或作为地址与数据信息标志等。发送时序如图3.13（a）所示。

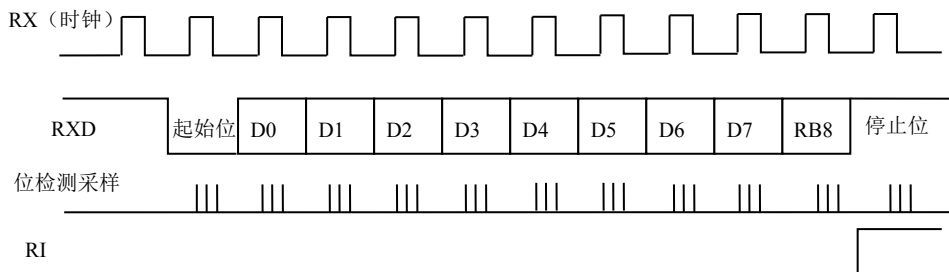
RXD作为数据的接收端，波特率固定：

$$\text{波特率} = \frac{2^{\text{SMOD}}}{64} \times f_{\text{osc}}$$

接收数据时，数据由RXD端输入，接收11位信息。当位检测逻辑采样到RXD引脚从1到0的负跳变，并判断起始位有效后，便开始接收1帧信息。接收时序如图3.13（b）所示。



（a）方式 2 发送时序



（b）方式 2 接收时序

图 3.13 方式 2 的发送和接收时序

（4）方式3（波特率可变11位异步通信方式）

除波特率可变外，引脚使用和数据格式同方式2，波特率的计算公式同方式1。

4. 波特率的计算

在串行通信中，收发双方对发送或接收数据的速率要有约定。通过软件可对单片机串行接口编程为四种工作方式，其中方式0和方式2的波特率是固定的，而方式1和方式3的波特率是可变的，由定时器T1的溢出率来决定。

串行接口的四种工作方式对应三种波特率。由于输入的移位时钟的来源不同，所以，

各种方式的波特率计算公式也不相同。

方式0的波特率= $f_{osc}/12$

方式2的波特率= $(2^{SMOD}/64) \times 2 \times f_{osc}$

方式1的波特率= $(2^{SMOD}/32) \times (T1 \text{ 溢出率})$

方式3的波特率= $(2^{SMOD}/32) \times (T1 \text{ 溢出率})$

当T1作为波特率发生器时，最典型的用法是使T1工作在自动再装入的8位定时器方式（即方式2，且TCON的TR1=1，以启动定时器）。这时溢出率取决于TH1中的计数值。

T1溢出率= $f_{osc} / \{12 \times [256 - (TH1)]\}$

在单片机的应用中，常用的晶振频率为：12 MHz和11.059 2 MHz。所以，选用的波特率也相对固定。常用的串行接口波特率以及各参数的关系如表3.6所示。

表 3.6 常用波特率与定时器 1 的参数关系

串口工作方式及波特率/(b/s)		f _{osc} / MHz	SMOD	定时器 T1		
				C/ *T	工作方式	初值
方式 1、3	62.5K	12	1	0	2	FFH
	19.2K	11.0592	1	0	2	FDH
	9600	11.0592	0	0	2	FDH
	4800	11.0592	0	0	2	FAH
	2400	11.0592	0	0	2	F4H
	1200	11.0592	0	0	2	E8H

3.2 学习实例

实例一 用 $\overline{INT0}$ 和 $\overline{INT1}$ 对按键进行计数并显示计数结果

1. 实例说明

要求在单片机的 $\overline{INT0}$ 和 $\overline{INT1}$ 引脚分别接两个按钮，通过按钮向单片机申请中断，每中断一次在中断服务程序中实现变量的加1和减1操作，并把结果送出显示（0~99）。注意C51中断函数的编写。如果用汇编程序要注意外部中断0和外部中断1的服务程序入口地址分别为0003H和0013H。

2. 仿真电路

用 $\overline{INT0}$ 和 $\overline{INT1}$ 对按键进行计数并显示计数结果仿真电路如图 3-14 所示。

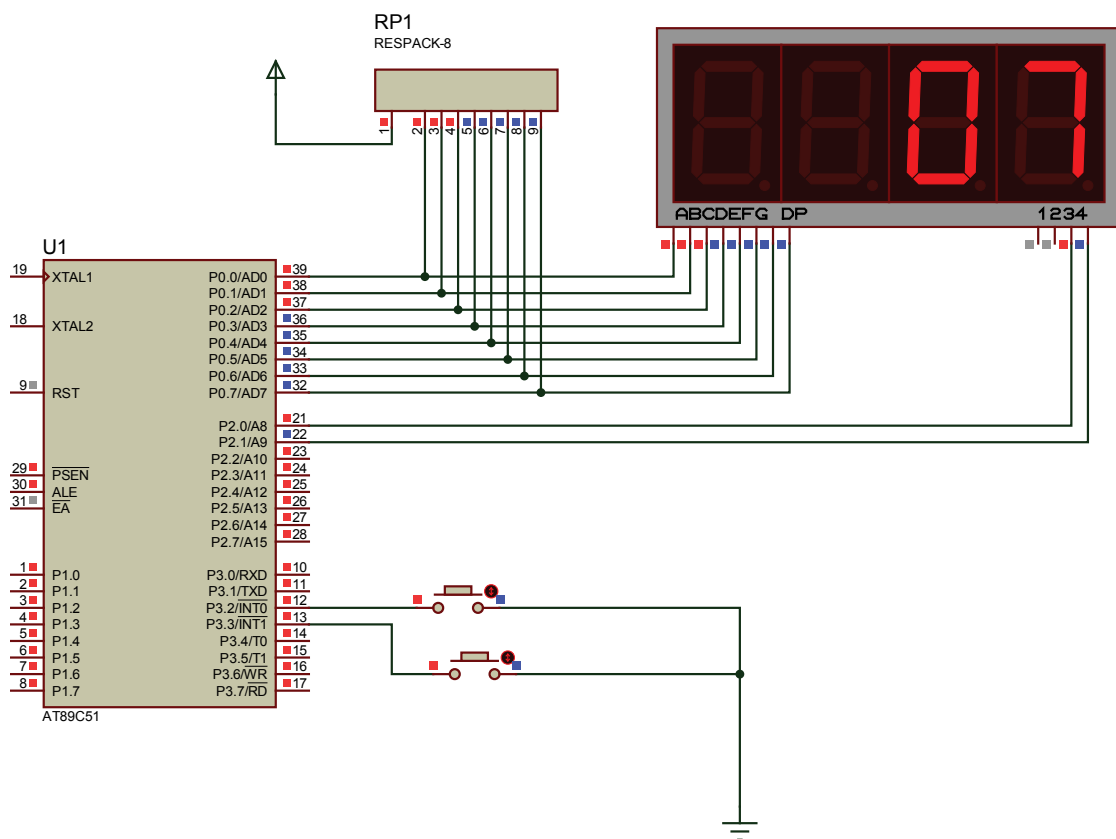


图 3.14 用INT0和INT1对按键进行计数并显示计数结果仿真电路

3. 程序设计

```
#include<at89x51.h> //头文件
unsigned char code segdata[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,
0x6f}; //共阴极 7 段显示器段码
char time;
void delay(void) //延时，用于 7 段显示器的动态刷新
{
    unsigned char i,j;
    for(i=4;i>0;i--)
        for(j=250;j>0;j--);
}
main()
{
    time=0; //变量初值
    EA=1; //开中断
    EX0=1; //允许 INT0 中断
    IT0=1; //跳沿触发
```

```
EX1=1; //允许 INT1 中断
IT1=1; //跳沿触发
while(1)
{
    P2=0xfd; //十位数位选
    P0=segdata[time/10]; //送十位数段码
    delay();
    P2=0xfe; //个位数位选
    P0=segdata[time%10]; //送个位数段码
    delay();
}

void int0(void) interrupt 0 using 0 //外部中断 0 中断函数
{
    time++;
    if(time>99)
    {
        time=0;
    }
}

void int1(void) interrupt 2 using 1 // //外部中断 1 中断函数
{
    time--;
    if(time<0)
    {
        time=0;
    }
}
```

实例二 用 T0 工作在方式 1 时控制播放一首歌曲

1. 实例说明

要求 T0 工作于方式 1，查资料了解音频控制、节拍控制的方法。

2. 仿真电路

用 T0 工作在方式 1 时控制播放一首歌曲仿真电路如图 3.15 所示。

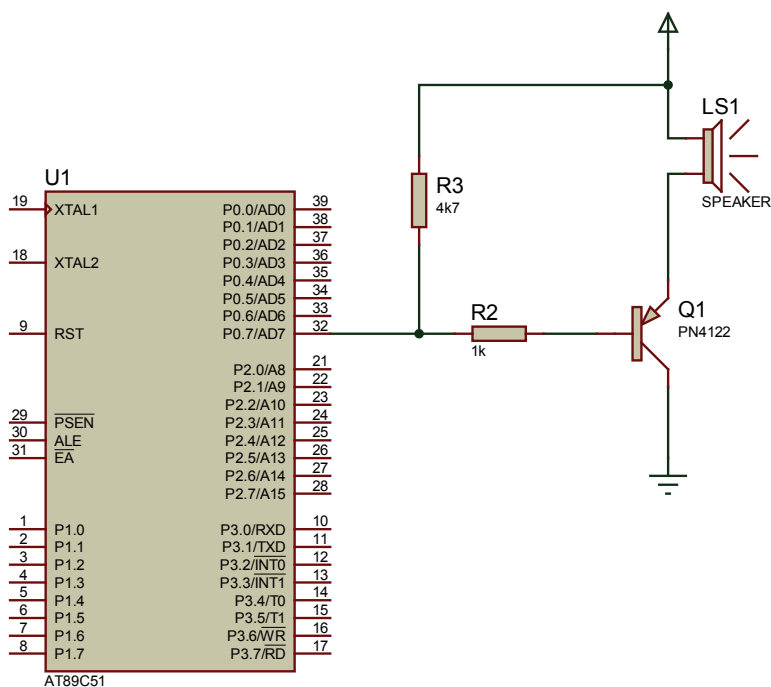


图 3.15 用 T0 工作在方式 1 时控制播放一首歌曲仿真电路

3. 程序设计

```

/*12Mhz 晶振工作*/
#include <reg51.h>
#define uint unsigned int
#define uchar unsigned char
sbit voice=P0^7;
uchar code sound[]={110,131,147,494,196,165,131,294,440,1,131,147,165,588,
196,440,294,660,330,165,196,880,588,220,262,124,110,196,220,330,131,147,495
,196,262,220,131,147,165,220,784,392,2,660,660,220,196,175,330,588,495,196,
110,131,147,2,131,147,330,392,440,524,247,220,196,165,880,880};
uchar zdjs=0, jp;
del(yj);
void main(void)
{
    uint dpjs=0;
    uchar yj;
    TMOD=0x01;
    IE=0x82;
    TH0=0xd8;
    TL0=0xef;
    TR0=1;
    while(1)
    {
        zdjs=0;
        dpjs++;
    }
}

```

```

yj=sound[dpjs];
dpjs++;
jp=sound[dpjs];
while(zdjs!=jp)
{
    if(yj!=0xff)
    {
        if(yj!=0)
        {
            voice=!voice;
            del(yj);
        }
        else
        {
            dpjs=0;
            break;
        }
    }
else
{
    voice=0;
    del(jp);
}
}
}
void time0() interrupt 1 using 1
{
    TH0=0xd8, TL0=0xef;
    zdjs++;
}
del(yj);
{
    uchar yj2=2;
    while(yj!=0)
    {
        while(yj2!=0)
        {
            yj2--;
        }
        yj2=2;
        yj--;
    }
}
}

```

实例三 用 T0 工作在方式 1 时控制 LED 灯的闪烁时间间隔

1. 实例说明

要求用 T0 实现 1 秒的定时，要注意长时间定时的方法，实现第一秒钟 1、3 亮，第二

秒钟 2、4 亮……

2. 仿真电路

用 T0 工作在方式 1 时控制 LED 灯的闪烁时间间隔仿真电路如图 3.16 所示。

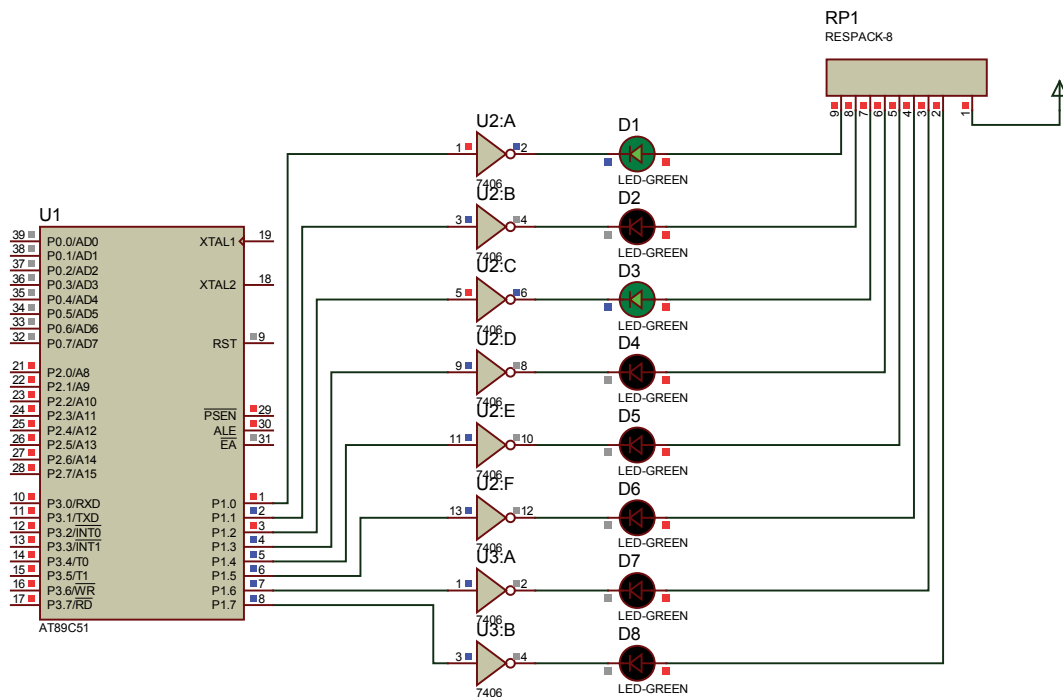


图 3.16 用 T0 工作在方式 1 时控制 LED 灯的闪烁时间间隔仿真电路

3. 程序设计

```
#include <reg51.h>
int a=0x0a;
int i=0;
int Tab[]={0x05,0x50,0x0a,0xa0,0x55,0xaa,0xff,0x00};
void main()
{
    TMOD=0x01;
    TH0=0x3c;
    TL0=0xb0;
    EA=1;
    ET0=1;
    TR0=1;
    while(1)
    {
    }
}
void int0() interrupt 1 using 0
{
    TR0=0;
```

```

a=a-1;
TH0=0x3c;
TL0=0xb0;
TR0=1;
if (a!=0)
{
    P1=Tab[i];
}
else
{
    a=0x0a;
    i=i+1;
}
if (i==8) i=0;
}

```

实例四 用 T0 工作在方式 2 时对脉冲进行计数并显示计数结果

1. 实例说明

要求用一个单片机的 P1.4 产生方波信号，用另一个单片机的 T0 工作在计数方式时对脉冲进行计数，并送七段显示器显示计数结果。

2. 仿真电路

用 T0 工作在方式 2 对脉冲进行计数并显示计数结果仿真电路如图 3.17 所示。

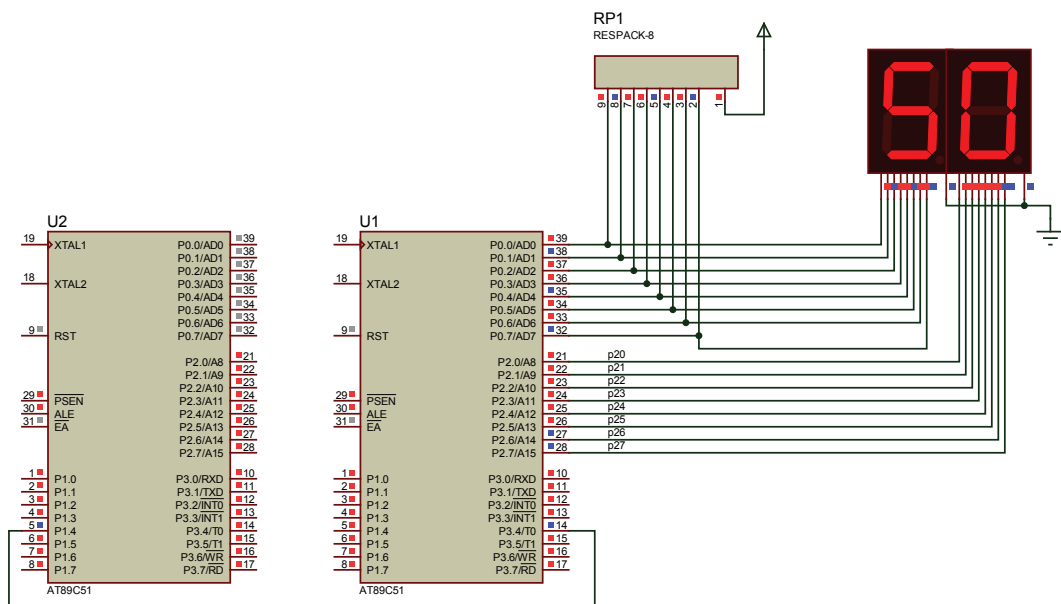


图 3.17 用 T0 工作在方式 2 时对脉冲进行计数并显示计数结果仿真电路

3. 程序设计

U2 产生方波的程序。

```

#include <reg51.h>
sbit fb=P1^4;
void delay(void)
{
    int m,n;
    for(m=0;m<100;m++)
        for(n=0;n<100;n++)
            ;
}

void main()
{
    int j;
    for(j=0;j<50;j++) //输出 50 个脉冲
    {
        fb=1;
        delay();
        fb=0;
        delay();
    }
    while(1)
    {

    }
}

```

U1 计数程序:

```

#include <reg51.h>
unsigned char code dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f,0x00};
void main()
{
    TMOD=0x06; //T0 工作方式 2 的计数方式
    TH0=0; //计数初值
    TL0=0;
    EA=0; //关中断
    TR0=1; //启动计数
    while(1)
    {
        P0=dispcode[TL0/10]; //十位数段码
        P2=dispcode[TL0%10]; //个位数段码
    }
}

```

实例五 用串行口工作在方式 0 时扩展输出接口

1. 实例说明

要求用串行口工作在方式 0 和 74LS164 扩展一个并行口,外接七段显示器循环显示 0~9 的数字。

2. 仿真电路

用串行口工作在方式 0 时扩展输出接口仿真电路如图 3.18 所示。

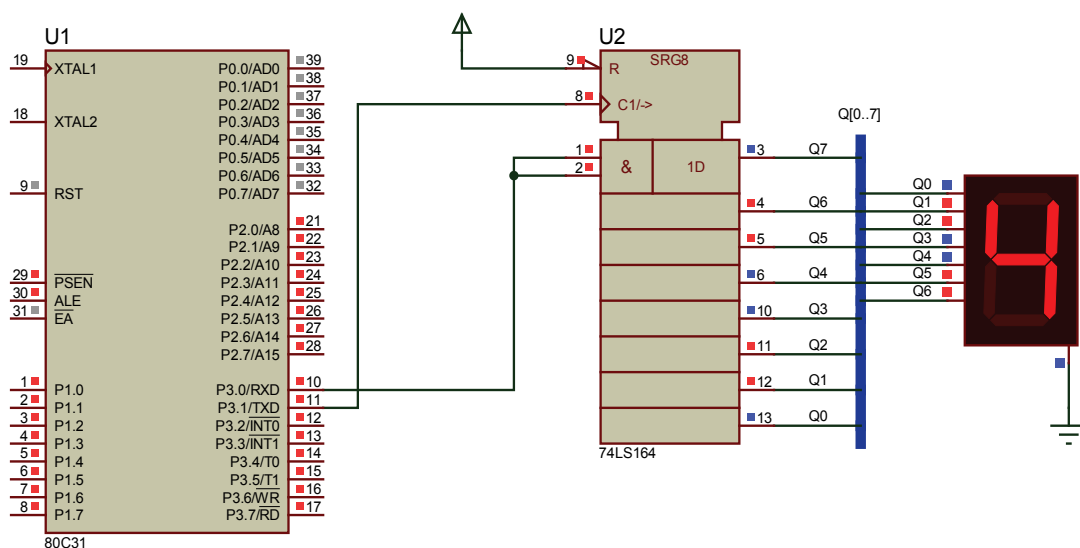


图 3.18 用串行口工作在方式 0 时扩展输出接口仿真电路

3. 程序设计

```
#include <reg52.h>
unsigned char code dispcode[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
0x7F,0x6F};
void delay(void)
{
    int m,n;
    for(m=0;m<400;m++)
        for(n=0;n<400;n++)
            ;
}
void main()
{
    int i=0;
    SCON=0x00;
    for(i=0;i<10;i++)
    {
        SBUF=dispcode[i];
        delay();
        while(TI==0)
            ;
        TI=0;
        if(i==10)
        {

```

```

        i=0;
    }

    }
}

```

实例六 用串行口工作在方式1时实现双机通信

1. 实例说明

要求单片机 U1 通过串口发送 0~9 的段码，单片机 U2 经过接收后通过 P1 口输出，送七段显示器显示。

2. 仿真电路

用串行口工作在方式1时实现双机通信仿真电路如图 3.19 所示。

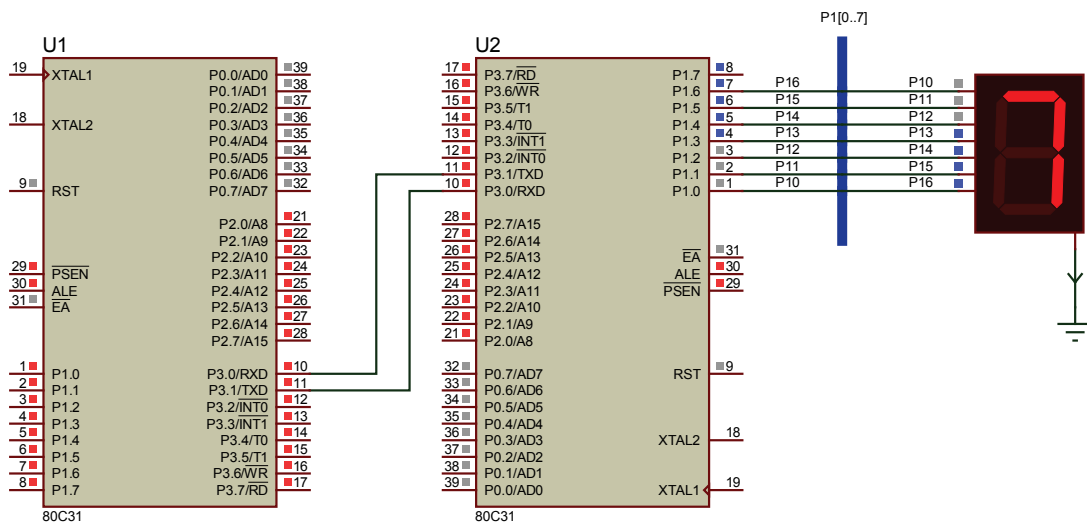


图 3.19 用串行口工作在方式1时实现双机通信仿真电路

3. 程序设计

```

//串行口发送程序
#include <reg51.h>
unsigned char code dispcode[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
0x7F,0x6F};
void delay(void)
{
    int m,n;
    for(m=0;m<400;m++)
        for(n=0;n<400;n++)
            ;
}

```

```

}
void main()
{
    int i;
    TMOD=0x20;    //定时器 T1 工作在方式 2 定时方式，波特率发生器
    TH1=0xfd;
    TL1=0xfd;
    TR1=1;
    SCON=0x40;    //串口工作在方式 1
    for(i=0;i<10;i++)
    {
        SBUF=dispcode[i];
        delay();
        while(TI==0);
        TI=0;
        if(i==10)
        {
            i=0;
        }
    }
}
//串行口接收程序
#include <reg51.h>
void main()
{
    TMOD=0x20;    //定时器 T1 工作在方式 2 定时方式，波特率发生器
    TH1=0xfd;
    TL1=0xfd;
    TR1=1;
    SCON=0x50;    //串口工作在方式 1，允许接收
    while(RI==0);
    RI=0;
    P1=SBUF;
}

```

实例七 单片机向 PC 机发送数据

1. 实例说明

要求通过串行接口向计算机发送 0~9 的数字，由计算机接收。调试时可用串口调试工具调试，仿真时可用 Proteus 的虚拟终端调试。仿真时省去了 MAX232，实际应用中需要电平转换，MAX232 的使用，请查阅相关手册，下一实例也省去了 MAX232。

2. 仿真电路

单片机向 PC 发送数据仿真电路如图 3.20 所示。

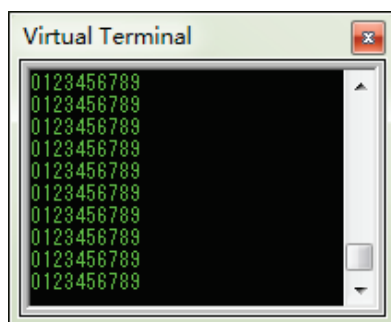
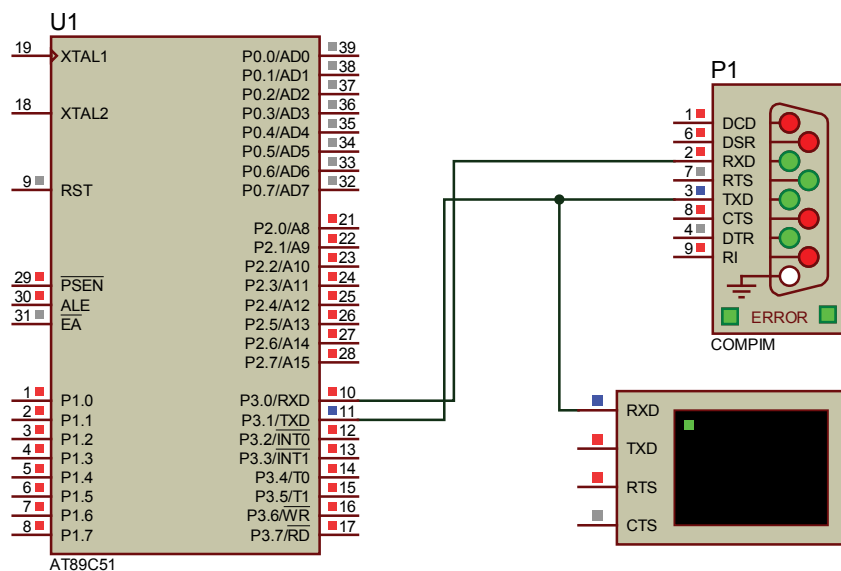


图 3.20 单片机向 PC 发送数据仿真电路

3. 程序设计

```
#include<reg51.h>
unsigned char code Tab[ ]={"0123456789"};
void ini(void)
{
    SCON=(SCON&0x2f)|0x50;
    PCON=0x80;
    TMOD=0x20;
    TH1=0xfa;
    TL1=0xfa;
    TR1=1;
    ES=0;
}
void send(char a)
{
    SBUF=a;
```

```
while(TI==0);
    TI=0;
}
void main(void)
{
    unsigned char i;
    ini();
    send(0x0c);
    while(1)
    {
        for(i=0;i<10;i++)    //模拟检测数据
        {
            send(Tab[i]);    //发送数据 i
        }
        send(0x0a);
        send(0x0d);
    }
}
```

实例八 单片机接收 PC 机发送的数据

1. 实例说明

要求由计算机发送 0~9 的段码，单片机通过串行口接收后由 P1 口输出，送七段显示器显示。

2. 仿真电路

单片机接收 PC 发送的数据仿真电路如图 3.21 所示。

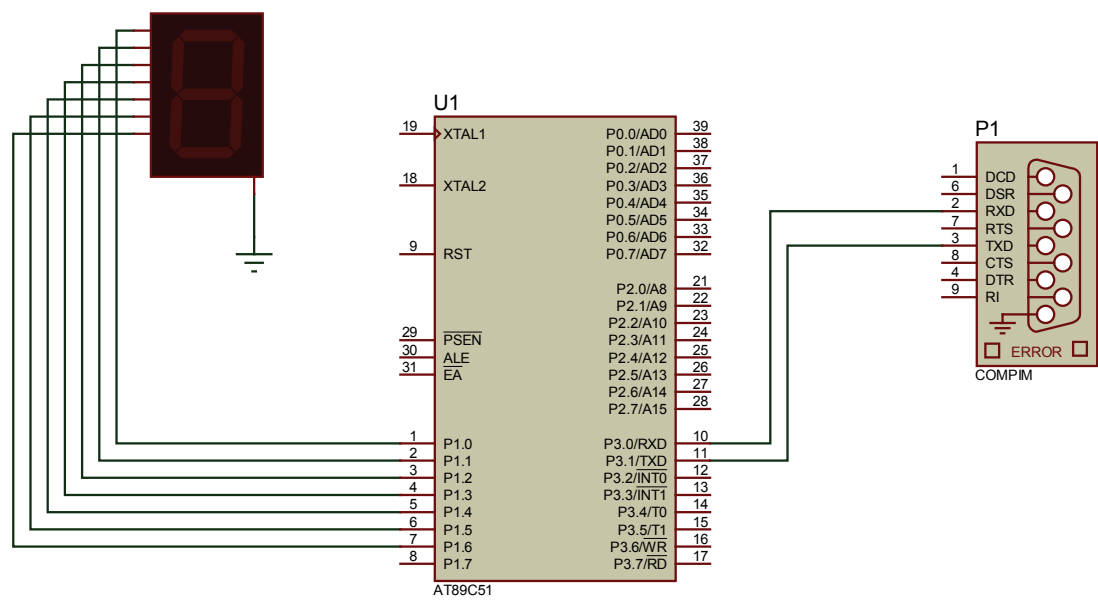


图 3.21 单片机接收 PC 发送的数据仿真电路

3. 程序设计

```
#include<reg51.h>
unsigned char dispcode[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,
0x6f};
char i=0;
char x;
void s() interrupt 4
{
    EA=0;
    RI=0;
    x=SBUF;
    if((x<='9')&&(x>='0'))
    {
        P1=dispcode[x-'0'];
    }
    else
    {
        P1=0x79;
    }
    EA=1;
}

void ini(void)
{
    SCON=(SCON&0x2f)|0x50;
    PCON=0x80;
    TMOD=0x20;
    TH1=0xfa;
    TL1=0xfa;
    TR1=1;
    ES=0;
}

void main(void)
{
    ini();
    SBUF=0x0c;
    P1=0;
    while(!TI);
    TI=0;
    EA=1;
    ES=1;
    while(1);
}
```

本章小结

通过本章的学习, 全面了解 51 单片机中断系统的结构、处理过程及其应用。掌握了 MCS-51 定时器/计数器 T0、T1 的结构、工作方式及其在实际工作中的应用, 就会更好地应用于实际。串行通信是工业控制网络的基础, 是构成复杂控制系统的关键。通过对串行口的学习, 为以后深入学习串行口的扩展及通信接口打下基础。

习题三

一、填空题

1. 外部中断请求标志位是_____和_____。
2. MCS-51 单片机外部中断请求信号有电平方式和_____, 在电平方式下, 当采集到 INT0、INT1 的有效信号为_____时, 激活外部中断。
3. 定时器 / 计数器的工作方式 3 指的是将_____拆成两个独立的 8 位计数器。而另一个定时器 / 计数器此时通常只可作为_____使用。
4. 8031 的异步通信口为_____ (单工/半双工/全双工)。
5. MCS-51 单片机片内有_____个中断源, 其中_____个外部中断源。
6. 设定 T1 为计数器方式, 工作方式 2, 则 TMOD 中的值为_____。

二、单项选择题

1. 控制串行口工作方式的寄存器是 ()。
A. TCON
B. PCON
C. SCON
D. TMOD
2. MCS-51 单片机的外部中断 1 的中断请求标志是 ()。
A. ET1
B. TF1
C. IT1
D. IE1
3. 在串行通信中, 8051 中发送和接收数据的寄存器是 ()。
A. SBUF
B. TMOD
C. SCON
D. DPTR
4. 用 MCS-51 串行接口扩展并行 I/O 口时, 串行接口工作方式应选择 ()。
A. 方式 0
B. 方式 1
C. 方式 2
D. 方式 3
5. 单片机的定时器/计数器设定为工作方式 0 时, 是 ()。
A. 8 位计数器结构
B. 2 个 8 位计数器结构
C. 13 位计数器结构
D. 16 位计数器结构
6. 外部中断源 IE1 (外部中断 1) 的向量地址为 ()。
A. 0003H
B. 000BH
C. 0013H
D. 002BH

7. 在 MCS-51 系统输出的异步串行数据流中, 必须确定的三个要素是 ()。
- A. 波特率、有效数据字长和奇偶校验
B. 波特率、有效数据字长和停止位数目
C. 波特率、同步或异步以及奇偶校验
D. 波特率、全双工或半双工以及奇偶校验
8. 在 MCS-51 中, 需要外加电路实现中断撤除的是 ()。
- A. 定时中断
B. 脉冲方式的外部中断
C. 外部串行中断
D. 电平方式的外部中断
9. 串行口工作方式 1 的波特率是 ()。
- A. 固定的, 为 $f_{osc}/32$
B. 固定的, 为 $f_{osc}/16$
C. 可变的, 通过定时器/计数器 T1 的溢出率设定
D. 固定的, 为 $f_{osc}/64$
10. 执行返回指令时, 返回的断点是 ()。
- A. 调用指令的首地址
B. 调用指令的末地址
C. 调用指令下一条指令的首地址
D. 返回指令的末地址
11. 各中断源发出的中断请求信号, 都会标记在 MCS-51 系统中的 ()。
- A. TMOD
B. TCON/SCON
C. IE
D. IP

三、编程题

1. 已知 80C51 单片机的 $f_{osc}=12\text{MHz}$, 用 T1 定时。试编程由 P1.0 和 P1.1 引脚分别输出周期为 2ms 和 500 μs 的方波 (要求用中断实现)。

2. 编一子程序, 从串行接口发送一个字符 '5' (波特率自己定)。

3. 应用定时器 T0 产生 1ms 的定时, 并使 P1.0 输出周期为 2ms 的方波, 设晶振频率为 6MHz, 要求工作在方式 1。

(1) 计算计数初值 X。

(2) 编出使 P1.0 输出周期为 2ms 的方波的程序段。

4. 按以下要求编写 8051 单片机定时器的初始化程序:

(1) T0 作为定时, 定时时间为 10ms。

(2) T1 作为计数, 记满 1000 溢出。

5. 采用定时/计数器 T0 对外部脉冲进行计数, 每计数 100 个脉冲, T0 切换为定时工作方式。定时 1ms 后, 又转为计数方式, 如此循环不止。假定 MCS-51 单片机的晶体振荡器的频率为 6MHz, 要求 T0 工作在方式 1, 请编写出相应程序。

6. 根据图 3.22 所示的电路图，按照如下要求进行程序设计：

(1) 利用 P3.2 ($\overline{\text{INT0}}$) 口控制 P1 口的 8 个 LED 灯。要求利用外部中断实现。

(2) 开始时 P1.0 上接的灯 D1 亮，外部中断 0 ($\overline{\text{INT0}}$) 接一个按钮，每中断一次，下一个 LED 灯亮，顺序下移，且每次只有一个 LED 灯亮，周而复始。

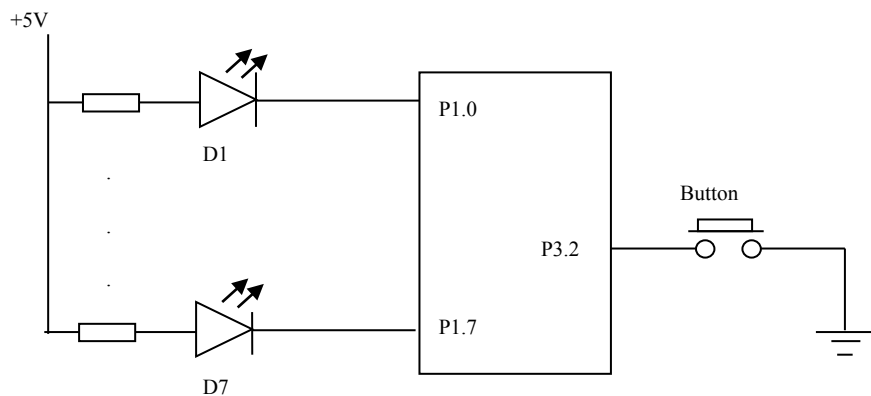


图 3.22 利用 P1 口控制 LED 灯的电路图



第4章 51 单片机系统扩展

学习目标

掌握 74LS373、74LS273、74LS244、74LS138、8279、8155、8255、74ls164、74ls165、AT24C02 等芯片的使用方法；理解 8051 单片机的总线扩展逻辑；掌握单片机存储器和 I/O 接口的扩展方法。

重点难点

单片机外部 RAM 的扩展,外部 ROM 的扩展,EEPROM 的扩展,键盘/显示接口的扩展。

4.1 知识结构

51 系列单片机芯片内部集成了 CPU、RAM、ROM、并行和串行 I/O 接口以及定时/计数器等，使用非常方便，对于小型应用系统已经足够了。但对于较大的应用系统，往往还需要扩展一些外围芯片（存储器或 I/O 口等），以弥补片内硬件资源的不足。

4.1.1 单片机系统总线及系统扩展方法

1. 单片机系统总线

单片机通过三总线（地址总线、数据总线和控制总线）扩展外部接口电路。图 4.1 是单片机的三总线结构示意图。

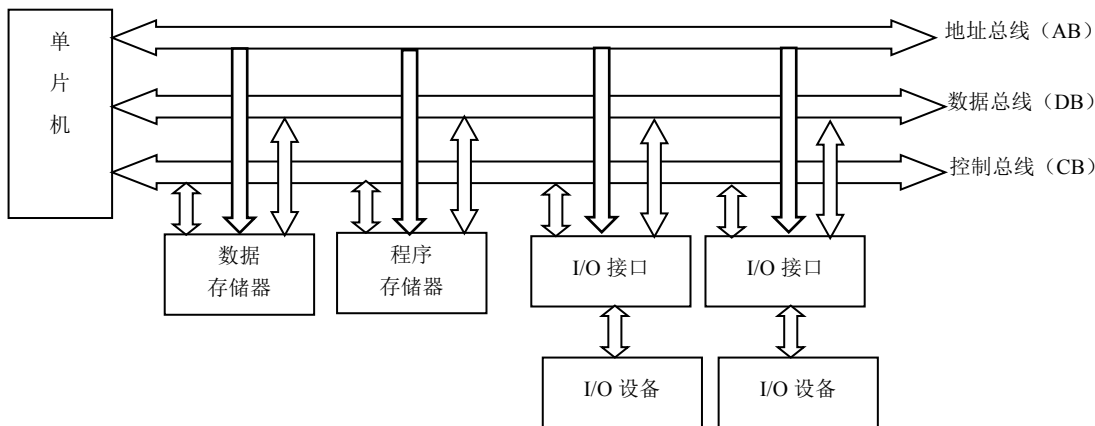


图 4.1 单片机三总线结构示意图

(1) 数据总线 (DB)

用于外围芯片和单片机之间进行数据传递, 比如将外部存储器中的数据送到单片机的内部, 或者将单片机中的数据送到外部的 D/A 转换器。在 51 单片机中, 数据的传递是用 8 根线同时进行的, 这 8 根线就被称之为数据总线。数据总线是双向的, 既可以由单片机传到外部芯片, 也可以由外部芯片传入单片机。

(2) 地址总线 (AB)

如果单片机扩展外部的存储器芯片, 在一个存储器芯片中有许多的存储单元, 要依靠地址进行区分, 在单片机和存储器芯片之间要用一些地址线相连。除存储器之外, 其他扩展芯片也有地址问题, 也需要和单片机之间用地址线连接, 各个外围芯片共同使用的地址线构成了地址总线。地址总线也是公用总线中的一种, 用于单片机向外部输出地址信号, 它是一种单向的总线。地址总线的根数决定了单片机可以访问的存储单元数量和 I/O 端口的数量。有 n 根线, 则可以产生 2^n 个地址编码, 访问 2^n 个地址单元。

(3) 控制总线 (CB)

这是一组控制信号线, 有一些是由单片机送出 (去控制其他芯片) 的, 而有一些则是由其他芯片送出 (由单片机接收以确认这些芯片的工作状态等) 的。对于 51 单片机而言, 这一类线的数量不多。这类线就其某一根而言是单向的, 可能是单片机送出的控制信号, 也可能是外部送到单片机的控制信号, 但就其总体而言, 则是双向的, 因为控制总线里面有几根是送出的, 有几根是接收的, 所以在图 4.1 中以双向的方式来表示控制总线。

2. 系统扩展的方法

通常和单片机接口连接的专用芯片也具备三总线引脚, 图 4.2 所示为各总线与单片机信号连接图。

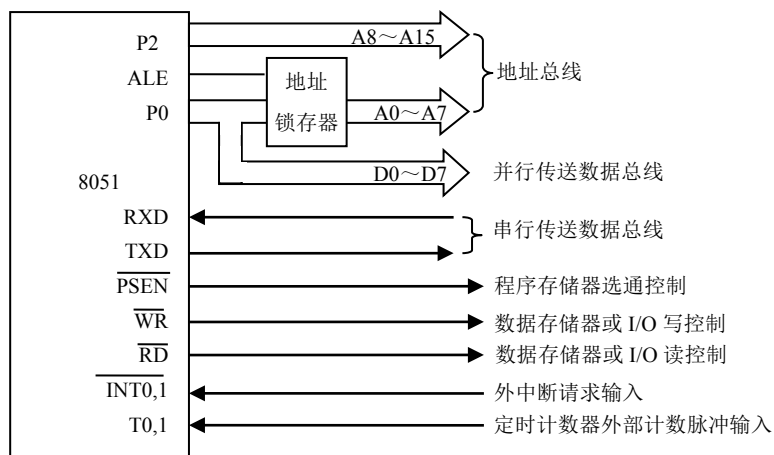


图 4.2 51 单片机总线接口信号图

P0 口为地址/数据线复用, 分时传送数据和低 8 位地址信息。在接口电路中, 通常配置地址锁存器 (如 74LS373、74LS573 等), 有 ALE 信号锁存低 8 位地址 A0~A7, 以分离地址和数据信息。

P2 口为高 8 位地址线，传送高 8 位地址 A8~A15。

$\overline{\text{PSEN}}$ 为程序存储器的控制信号，在取指令码时或执行 MOV_C 指令时变为有效； $\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ 为数据存储器 and I/O 口的读、写控制信号，在执行 MOV_X 指令时变为有效。

系统的扩展归结为三总线的连接，连线时应遵守下列原则：

(1) 数据线的连接

外接芯片的数据线 D0~D7 与单片机的数据线 D0~D7 (P0 口) 连接。

(2) 控制线的连接

控制总线由 $\overline{\text{WR}}$ 、 $\overline{\text{RD}}$ 、 $\overline{\text{EA}}$ 、ALE 和 $\overline{\text{PSEN}}$ 等信号组成，用于读/写控制、片外 ROM 选通、地址锁存控制和片内、片外 ROM 选择。

(3) 地址线的连接

地址总线宽度为 16 位，可寻址范围达 2^{16} ，即 64 K。低 8 位 A7~A0 由 P0 接口经地址锁存器提供，高 8 位 A15~A8 由 P2 接口提供。由于 P0 接口是数据、地址分时复用的，所以 P0 接口输出的低 8 位地址必须用地地址锁存器进行锁存。

3. 存储器扩展的编址技术

进行存储器扩展时，可供使用的编址方法有两种，即：线选法和译码法。

(1) 线选法

所谓线选法，就是直接以系统的地址线作为存储芯片的片选信号，为此只需把高位地址线与存储芯片的片选信号直接连接即可。特点是简单明了，不需增加另外电路。缺点是存储空间不连续。适用于小规模单片机系统的存储器扩展。

(2) 译码法

所谓译码法就是使用译码器对系统的高位地址进行译码，以其译码输出作为存储芯片的片选信号。这是一种最常用的存储器编址方法，能有效地利用空间，特点是存储空间连续，适用于大容量多芯片存储器扩展。

常用的译码芯片有 74LS139 和 74LS138 等，它们的 CMOS 型芯片分别是 74HC139 和 74HC138。74LS139 是双 2~4 译码器，即对 2 个输入信号进行译码，得到 4 个输出状态。74LS138 是 3~8 译码器，即对 3 个输入信号进行译码，得到 8 个输出状态。

74LS138 引脚如图 4.3 所示，译码功能如表 4.1 所示。

当一个选通端 (G1) 为高电平，另两个选通端 ($\overline{\text{G2A}}$ 和 $\overline{\text{G2B}}$) 为低电平时，可将地址端 (A、B、C) 的二进制编码在一个对应的输出端以低电平译出。

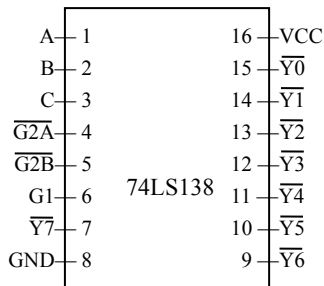


图 4.3 74LS138 引脚图

表 4.1 74LS138 译码功能

输 入						输 出							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	$\overline{Y7}$	$\overline{Y6}$	$\overline{Y5}$	$\overline{Y4}$	$\overline{Y3}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
其他状态			x	x	x	1	1	1	1	1	1	1	1

74LS139 为双 2 线~4 线译码器，这两个译码器完全独立，分别有各自的数据输入端、译码状态输出端以及数据输入允许端，也可作数据分配器。引脚如图 4.4 所示，译码功能如表 4.2 所示。

当选通端（G1）为高电平，可将地址端（A、B）的二进制编码在一个对应的输出端以低电平译出。若将选通端（G1）作为数据输入端时，139 还可作数据分配器。

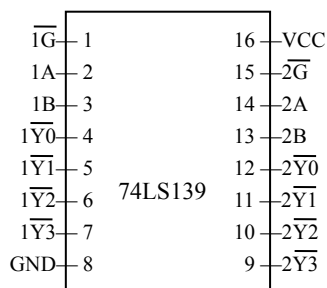


图 4.4 74LS139 引脚图

表 4.2 74LS139 译码功能

输 入			输 出			
允 许	选 择					
\overline{G}	B	A	$\overline{Y3}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$
1			1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

4.1.2 单片机存储器的扩展

1. 程序存储器的扩展

当程序较长、片内程序存储器容量不够时，必须在单片机外部扩展程序存储器。MCS-51

单片机片外有 16 条地址线,即 P0 口和 P2 口,因此最大寻址范围为 64 KB(0000H~FFFFH)。

MCS-51 单片机有一个管脚 $\overline{\text{EA}}$ 跟程序存储器的扩展有关。如果接高电平,那么片内存储器地址范围是 0000H~0FFFH (4 KB),片外程序存储器地址范围是 1000H~FFFFH (60 KB)。如果接低电平,不使用片内程序存储器,片外程序存储器地址范围为 0000H~FFFFH (64 KB)。

(1) 程序存储器典型芯片

程序存储器需具有系统掉电后信息不会丢失的特性,因此,EPROM、EEPROM、Flash 芯片都可以作为程序存储器。EPROM 是紫外线擦除的程序存储器,其典型芯片是 27 系列产品,如 2716、2764、27C020(8*256K)等,程序修改不太方便,在一些需要系统有在线编程功能时,就只能用 EEPROM 和 Flash 作为程序存储器。

电擦除可编程只读存储器 EEPROM 是一种可用电气方法在线擦除和再编程的只读存储器,它既有 RAM 可读可改写的特性,又具有长期非易失地保存信息的优点。因此,EEPROM 在单片机存储器扩展中,可以用作程序存储器。

EEPROM 作为程序存储器使用时,CPU 读取 EEPROM 数据同读取一般 EPROM 操作相同;但 EEPROM 的写入时间较长,必须用软件或硬件来检测写入周期。

常用的 EEPROM 芯片如表 4.3 所示,它们有如下特点:

单+5 V 供电,电可擦除可改写,使用次数为 1 万次,信息保存时间为 10 年,读出时间为 ns 级,写入时间为 ms 级。芯片引脚信号与相应的 RAM 和 EPROM 芯片兼容,见表 4.3 所列。

表 4.3 常用 EEPROM 芯片

型 号	引 脚 数	容量/字节	引脚兼容的存储器
2816	24	2 KB	2716, 6116
2817	28	2 KB	
2864	28	8 KB	2764, 6264
28C256	32	32 KB	27C256
28F512	32	64 KB	27C512
28F010	32	128 KB	27C010
28F020	32	256 KB	27C020
28F040	32	512 KB	27C040

(2) 地址锁存器 74LS373 芯片

地址锁存器可使用 74LS373 或 74LS573 (两者性能一样,只是后者引脚排列便于印制板的设计),它们实质上是一个带三态缓冲输出的 8D 触发器。74LS373 引脚如图 4.5 所示。

D7~D0 为 8 位数据输入线;Q7~Q0 为 8 位数据输出线;G 为数据输入锁存选通引脚,高电平有效,当该信号为高电平时,外部数据选通到内部锁存器,负跳变时,数据锁存到锁存器中;/OE 为数据输出允许引脚,低电平有效,当该信号为低电平时,三态门打开,锁存器数据输出到数据输出线,当该信号为高电平时,输出线为高阻态。

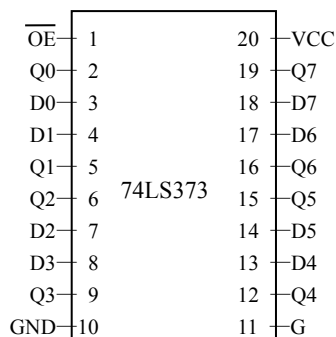


图 4.5 74LS373 引脚图

(3) EPROM 27256

27256 是 32K*8 字节的紫外线擦除、可编程的只读存储器，单一+5V 供电，工作电流为 100mA，维持电流为 40mA，读出时间最大为 250ns，28 脚双列直插式封装。引脚如图 4.6 所示，各引脚的含义为：

A0~A14 为 15 根地址线，可寻址 32KB 字节；D0~D7 为数据输出线；CE 为片选线；OE/VPP 为数据输出选通线/编程电源。

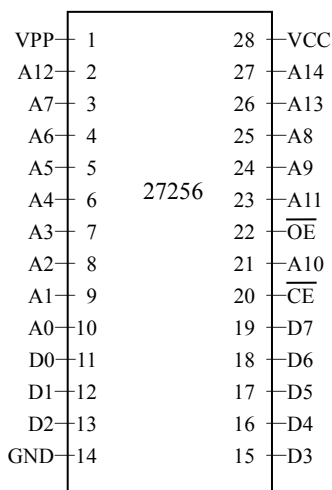


图 4.6 27256 引脚图

(4) EEPROM 2864

2864(2864A)为 8KB 字节 EEPROM，维持电流为 60mA，典型读出时间为 200~350ns，字节编程写入时间为 10~20ms，芯片内有电压提升电路，编程是不必增加电压，单一+5 V 供电即可。其引脚如图 4.7 所示，引脚和 6264、2764 兼容。操作方式及 I/O 引脚状态如表 4.4 所示。

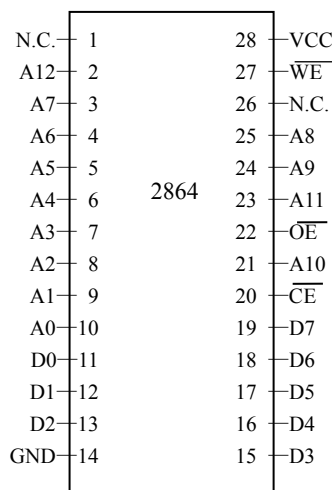


图 4.7 2864 的引脚

表 4.4 2864 的操作方式及 I/O 引脚状态

方式 \ 功能端	$\overline{\text{CE}}$ (20)	$\overline{\text{OE}}$ (22)	$\overline{\text{WE}}$ (27)	D0~D7
维持	高	x	x	高阻抗
读	低	低	高	数据输出
写	低	高	低	数据输入
数据查询	低	低	高	数据输出

(5) EEPROM AT24C02

AT24C02 是一个 2KB 位串行 CMOS EEPROM, 内部含有 256 个 8 位字节, CATALYST 公司的先进 CMOS 技术实质上减少了器件的功耗。AT24C02 有一个 16 字节页写缓冲器。该器件通过 I²C 总线接口进行操作, 有一个专门的写保护功能。其引脚如图 4.8 所示。

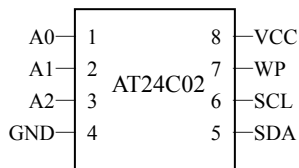


图 4.8 AT24C02 引脚

各引脚的含义为:

A0、A1、A2 为器件地址选择引脚, 这些输入脚用于多个器件级联时设置器件地址, 当这些脚悬空时默认值为 0。当使用 AT24C02 时最大可级联 8 个器件。如果只有一个 AT24C02 被总线寻址这三个地址输入脚 (A0、A1、A2) 必须连接到 GND。

SDA 为串行数据、地址引脚, AT24C02 双向串行数据/地址管脚用于器件所有数据的发送或接收, SDA 是一个开漏输出管脚, 可与其他开漏输出或集电极开路输出进行线或 (wire-OR)。

SCL 为串行时钟, AT24C02 串行时钟输入管脚用于产生器件所有数据发送或接收的时钟, 这是一个输入管脚。

WP 为写保护信号, 如果 WP 管脚连接到 VCC, 所有的内容都被写保护只能读。当 WP 管脚连接到 GND 或悬空允许器件进行正常的读/写操作。

VCC 为+1.8~6.0V 工作电压, GND 接地。

2. 数据存储器的扩展

MCS-51 单片机内只有 128 字节的数据 RAM, 当单片机用于实时数据采集或处理大批量数据时, 仅靠片内提供的 RAM 是远远不够的, 此时只能在片外扩展。RAM 有 DRAM (动态存储器) 和 SRAM (静态存储器), DRAM 需定时充电刷新, 单片机中不采用。

单片机系统中常用的 SRAM 芯片的典型型号有: 6116 (2K×8), 6264 (8K×8), 62128 (16K×8), 62256 (32K×8)。它们都用单一+5V 电压供电, 双列直插封装, 6116 为 24 引脚封装, 6264、62128、62256 为 28 引脚封装。62256 的引脚图如图 4.9 所示。

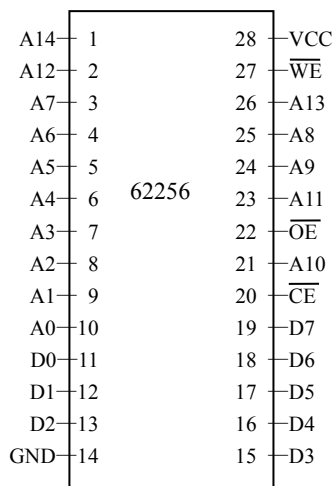


图 4.9 62256 引脚图

图中的各 RAM 引脚功能如下:

A0~A14: 地址输入线。

D0~D7: 双向三态数据线。

\overline{CE} : 片选信号输入线, 低电平有效。对于 6264 芯片, 当 24 脚(CS)为高电平且 \overline{CE} 为低电平时才选中该片。

\overline{OE} : 读选通信号输入线, 低电平有效。

\overline{WE} : 写允许信号输入线, 低电平有效。

VCC: 工作电源+5 V。

GND: 地。

静态 SRAM 存储器有读出、写入、维持三种工作方式, 这些工作方式的操作控制如表 4.5 所示。

表 4.5 6116、6264、62256 的操作方式

方式 \ 信号	$\overline{\text{CE}}$	$\overline{\text{OE}}$	$\overline{\text{WE}}$	D0~D7
读	低	低	高	数据输出
写	低	低	低	数据输入
维持	高	x	x	高阻抗

4.1.3 并行 I/O 口扩展

在无片外存储器扩展的系统中，51 单片机的 4 个端口都可以作为准双向通用 I/O 口使用。而在具有片外扩展存储器的系统中，P0 口分时地作为低 8 位地址线 and 数据线，P2 口作为高 8 位地址线。这时，P0 口和部分或全部的 P2 口无法再做通用 I/O 口。P3 口具有第二功能，在应用系统中也常被使用。因此在大多数的应用系统中，真正能够提供给用户使用的只有 P1 和部分 P2、P3 口。所以，51 单片机的 I/O 端口通常需要扩充，以便和更多的外设进行联系。在 51 单片机中扩展的 I/O 口采用与片外数据存储器相同的寻址方法，所有扩展的 I/O 口，以及通过扩展 I/O 口连接的外设都与片外 RAM 统一编址。

1. 采用 TTL 电路扩展

TTL 电路扩展 I/O 口是一种简单的 I/O 口扩展方法。它具有电路简单、成本低、配置灵活的特点。

(1) 74LS244 芯片介绍

74LS244 为 8 缓冲线驱动器（三态输出）， $\overline{\text{1G}}$ 和 $\overline{\text{2G}}$ 为低电平有效的使能端。当二者之一为高电平时，输出为三态。引脚图如图 4.10 所示。

(2) 74LS273 芯片介绍

74LS273 为 8D 触发器， $\overline{\text{MR}}$ 为低电平有效的清除端。当 $\overline{\text{MR}}=0$ 时，输出全为 0 且与其他输入端无关；CLK 端是时钟信号，当 CLK 由低电平向高电平跳变时刻，D 端输入数据传送到 Q 输出端。引脚图如图 4.11 所示。

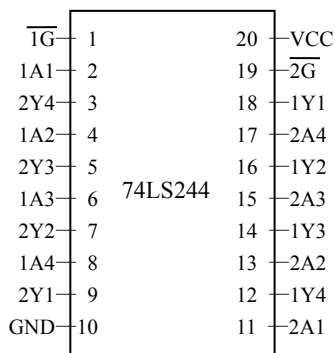


图 4.10 74LS244 引脚图

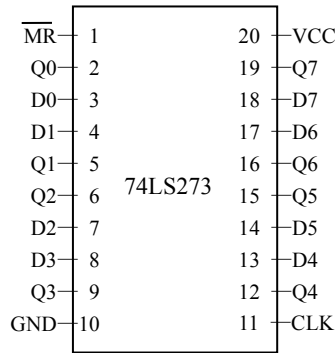


图 4.11 74LS273 引脚图

2. 采用 8255 芯片扩展 I/O 接口

8255 是 Intel 公司生产的可编程并行 I/O 接口芯片，有 3 个 8 位并行 I/O 口，具有 3 个通道，3 种工作方式的可编程并行接口芯片（40 引脚）。其各口功能可由软件选择，使用灵活，通用性强。8255 可作为单片机与多种外设连接时的中间接口电路。

（1）8255 内部结构和引脚功能介绍

8255 共有 40 个引脚，采用双列直插式封装，引脚如图 4.12 所示，内部结构如图 4.13 所示，各引脚功能如下：

D7~D0：三态双向数据线，与单片机数据总线连接，用来传送数据信息。

$\overline{\text{CS}}$ ：片选信号线，低电平有效。

$\overline{\text{RD}}$ ：读出信号线，低电平有效，控制数据的读出。

$\overline{\text{WR}}$ ：写入信号线，低电平有效，控制数据的写入。

VCC：+5V 电源。

PA7~PA0：A 口输入/输出线。

PB7~PB0：B 口输入/输出线。

PC7~PC0：C 口输入/输出线。

RESET：复位信号线。

A1~A0：地址线，用来选择内部端口。

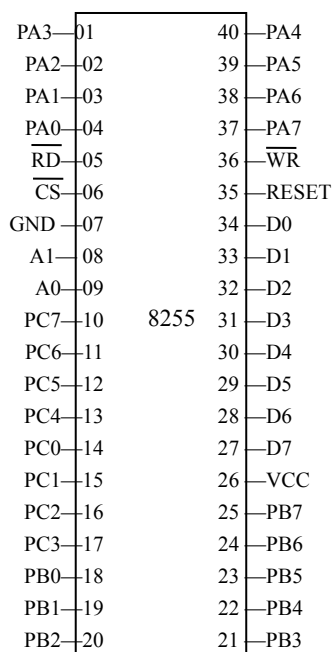


图 4.12 8255 引脚图

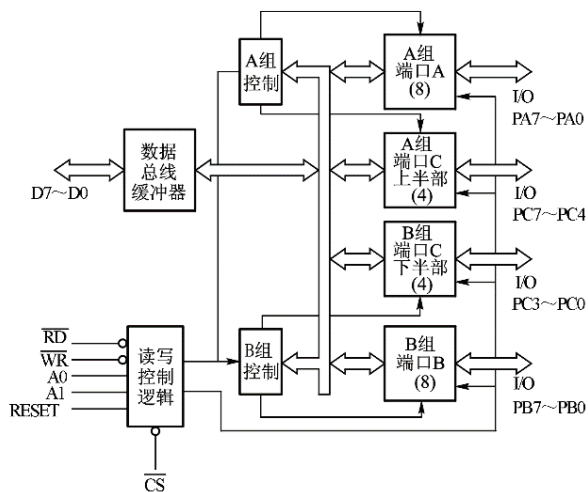


图 4.13 8255 内部结构图

（2）8255 的工作方式

8255 有 3 种工作方式，这些工作方式可用工作方式控制字来指定，如表 4.6 所示。

表 4.6 8255 工作方式

方 式	A	B	C
方式 0	基本 I/O 方式	基本 I/O 方式	基本 I/O 方式
方式 1	应答 I/O 方式	应答 I/O 方式	通信线
方式 2	双向应答 I/O 方式	无	通信线

(3) 8255 的控制字

8255A 有两个控制字，即方式控制字和 C 口置位/复位控制字，这两个控制字公用一个地址通过最高位来选择使用哪个控制字。

① 工作方式选择字

8255 工作方式选择字共 8 位，如图 4.14 所示，存放在 8255 控制寄存器中，最高为 D7 为标志位，D7=1 表示控制寄存器中存放的是工作方式选择字，D7=0 表示控制寄存器中存放的是 C 口置位/复位控制字。

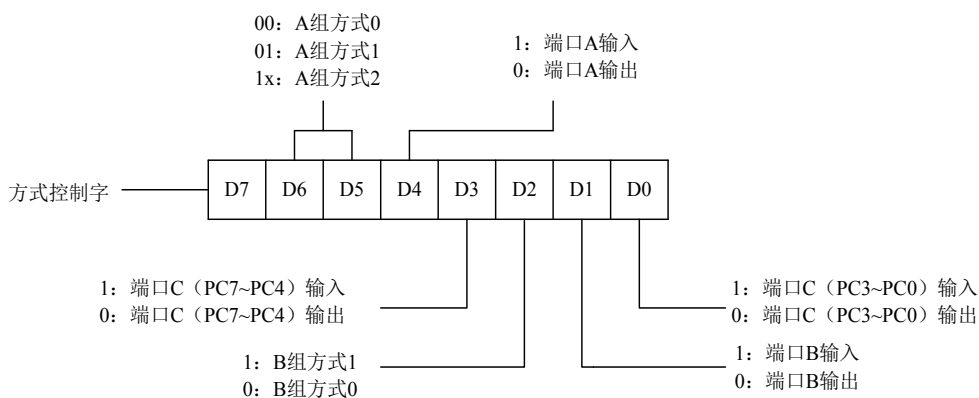


图 4.14 8255 工作方式控制字

D3~D6 用于 A 组的控制；D0~D2 用于 B 组的控制。

② C 口置位/清零控制字

8255 的 C 口可进行位操作，即可对 8255 的 C 口的每一位清零操作，该操作通过设置 C 口置位/复位来实现，各位含义如图 4.15 所示。

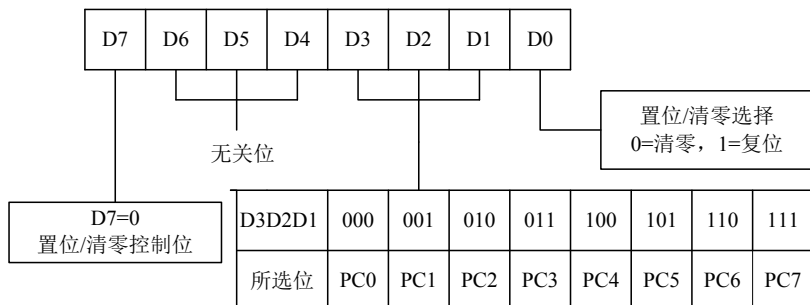


图 4.15 C 口置位/清零控制字

由于 8255 的工作方式选择字与 C 口置位/复位公用一个控制寄存器, 故设 D7 为标志位, D7=0 表示控制字为 C 口置位/复位。

3. 扩展多功能接口芯片 8155

8155 片内具有 256 字节的静态 RAM、两个 8 位和 1 个 6 位的可编程并行 I/O 接口、一个 14 位的有多重工作方式的减法计数器, 以及一个地址锁存器。51 单片机外接一片 8155 后, 就综合地扩展了数据 RAM、I/O 接口和定时/计数器。

(1) 8155 的内部结构图及芯片引脚功能设置

8155 有 40 个引脚, 采用双列直插封装, 其引脚图和组成内部结构方框图如图 4.16 所示。

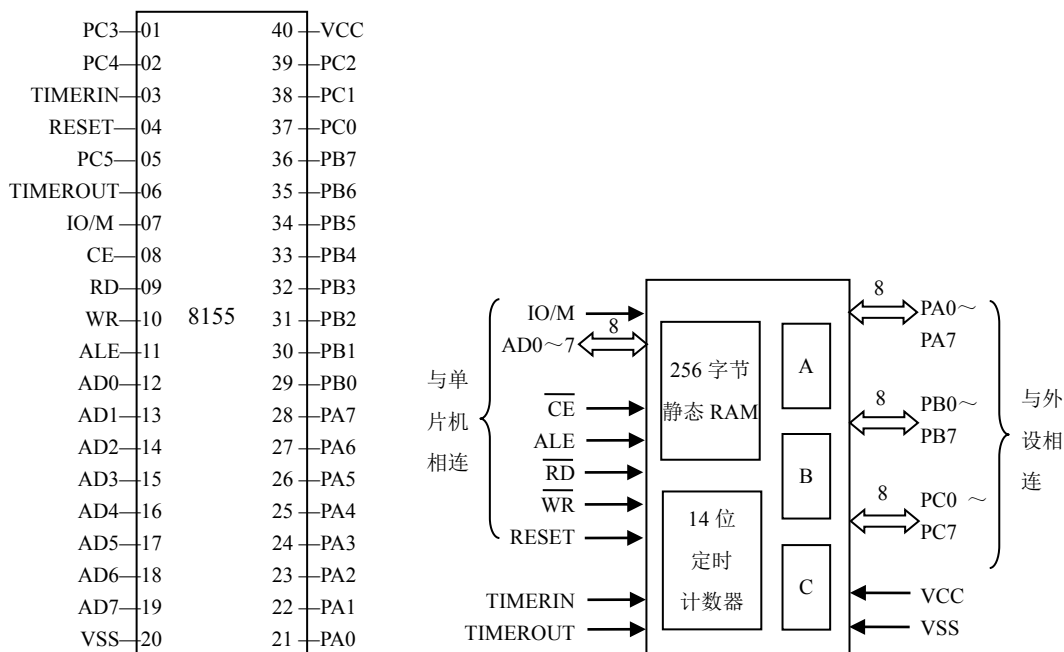


图 4.16 8155 引脚图和内部结构方框图

图中：

AD0~AD7 为三态地址/数据线 (8 条)。

PA0~PA7、PB0~PB7、PC0~PC5 为 I/O 口总线 (22 条)。

控制总线 (8 条)

ALE：地址锁存 (输入)；IO/ \overline{M} ：I/O 口/RAM 选择, 为 0 时选内 RAM, 为 1 时选内 I/O 口； \overline{CE} ：片选线； \overline{RD} 、 \overline{WR} ：读、写控制；TIMERIN：定时器输入 (输入定时器所需时钟)；TIMEROUT：定时器输出 (输出所产生的方波脉冲)。

8155 的 RAM 和 I/O 口地址分配如下表 4.7 所示。

表 4.7 8155 端口地址分配

IO/ \overline{M}	AD2	AD1	AD0	D0~D7
1 (I/O 口)	0	0	0	命令/状态口
	0	0	1	PA 口
	0	1	0	PB 口
	0	1	1	PC 口
	1	0	0	TIME 低 8 位 TL
	1	0	1	TIME 高 8 位 TH
0 (存储器)	AD0~AD7			内部 RAM

(2) 8155 的命令控制字

8155 有一个控制命令寄存器和一个状态标志寄存器。8155 的工作方式由 CPU 写入命令寄存器的命令字来确定。命令寄存器只能写入不能读出，命令字的格式如表 4.8 所示。

表 4.8 命令字的格式

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
TM ₂	TM ₁	IE _B	IE _A	PII	PI	P _B	P _A
TIMER 工作方式		B 口中断允/禁	A 口中断允/禁	I/O 端口工作方式		B 口 I/O	A 口 I/O

P_B 和 P_A 分别用来选择 A 口和 B 口的输入和输出方式：置 1，选择输出方式；置 0，选择输入方式。IE_A 和 IE_B 分别用来选择 A 口和 B 口的中断允许和禁止状态：置 1，选择中断允许；置 0，选择中断禁止。

PI 和 PII 用来选择并行口的工作方式：

PIIPI=00：A 口和 B 口为基本的 I/O 方式，C 口为输入方式；

PIIPI=11：A 口和 B 口为基本的 I/O 方式，C 口为输出方式；

PIIPI=01：B 口为基本 I/O 方式，A 口为选通 I/O 方式，PC₂~PC₀ 为 A 口的联络信号；

PIIPI=10：A 口、B 口为选通 I/O 方式，C 口为联络信号。

当 A 口、B 口工作为选通 I/O 方式，C 口为联络信号，各位意义如下：

PC₀：INTR_A，PC₁：BF_A，PC₂：STB_A，PC₃：INTR_B，PC₄：BF_B，PC₅：STB_B。

其中，STB 为选通信号，BF 为缓冲器满信号，INTR 为中断请求信号。

TM₂ 和 TM₁ 控制定时/计数器的工作方式：

TM₂TM₁=00，不影响计数器工作；TM₂TM₁=01，停止计数器工作；

TM₂TM₁=10，计数器回零，停止工作；TM₂TM₁=11，启动计数器工作。

(3) 8155 的状态字

8155 的状态字用于查询或检测中断状态，格式如表 4.9 所示。

表 4.9 查询或检测中断状态格式

X	TIMER	INTE _B	BF _B	INTR _B	INTE _A	BF _A	INTR _A
不用	计数满/未滿	B 口中断允/禁	B 口缓冲器满/空	B 口中断有无	A 口中断允/禁	A 口缓冲器满/空	A 口中断有无

TIMER：在读状态字后或硬件复位后为 0，有定时器溢出中断发生为 1；

INTE_A、INTE_B 中断允许为 1，中断禁止为 0；

BF_A、BF_B 缓冲器空为 0，满为 1；INTR_A、INTR_B：有中断请求为 1，无中断请求为 0。

(4) 内部定时器/计数器及使用

8155 的可编程定时/计数器是一个 14 位的减法计数器，在 TIMERIN 端输入计数脉冲，计满时由 TIMEROUT 输出脉冲或方波，输出方式由定时高 8 位寄存器中的 M2、M1 两位来决定。当 TIMERIN 接外脉冲时为计数方式，接系统时钟时为定时方式，实际使用时一定要注意芯片允许的最高计数频率。

定时/计数器的初始值和输出方式由高、低 8 位寄存器的内容决定，初始值 14 位，其余 2 位定义输出方式。其中低 8 位寄存器存放计数初始值的低 8 位，高 8 位寄存器的格式如下：


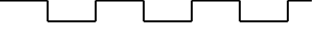


M2	M1						
----	----	--	--	--	--	--	--

输出方式 计数初始值高 6 位

① 定时/计数器的输出方式。

定时器的输出方式见表 4.10 所列。

表 4.10 定时器的输出方式

M2	M1	方 式	波 形
0	0	在一个计数周期输出单次方波连续方波	
0	1	在计满回 0 后输出的单个脉冲连续脉冲	
1	0		
1	1		

② 定时/计数器的工作。

8155 对内部定时器的控制是由 8155 控制字的 D7、D6 位决定的，其工作情况总结如表 4.11 所列。

表 4.11 计数器的工作情况

8155 的控制字		定时/计数器工作情况
D7	D6	
0	0	无操作，即不影响定时器的的工作
0	1	立即停止定时器的计数
1	0	定时器计满回 0 后停止技术
1	1	开始计数：若定时器不工作，则开始计数；若定时器正在计数，则计满回 0 后按新输入的长度值开始计数

4. 8279 扩展芯片介绍

INTEL 8279 是一种可编程键盘/显示器接口芯片，它含有键盘输入和显示器输出两种功能。键盘输入时，它提供自动扫描，能与按键或传感器组成的矩阵相连，接收输入信息，它能自动消除开关抖动并能对多键同时按下提供保护。显示输出时，它有一个 16×8 位显示 RAM，其内容通过自动扫描，可由 8 位或 16 位 LED 数码管显示。

(1) 8279 的引脚排列和功能

8279 采用 40 引脚双列直插封装，其引脚排列及功能方框图分别如图 4.17 (a)、(b) 所示。

其引脚功能如下：

D0~D7：数据总线，双向三态总线。

CLK：系统时钟输入端。

RESET：系统复位输入端，高电平有效，复位状态为：16 个字符显示；编码扫描键盘——双键锁定；程序时钟编程为 31。

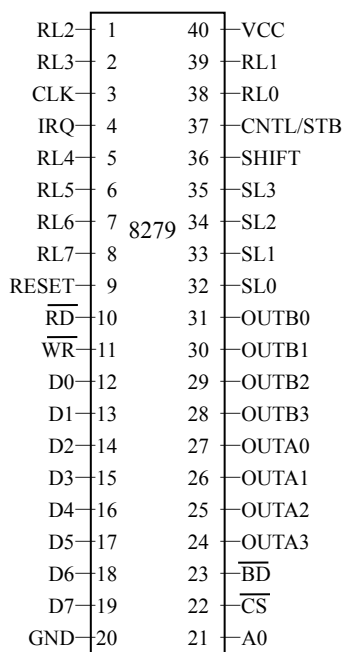
\overline{CS} ：片选输入端，低电平有效。

A0：数据选择输入端，A0=1 时，CPU 写入数据为命令字，读出状态字为状态字；A0=0 时，CPU 读、写均为数据。

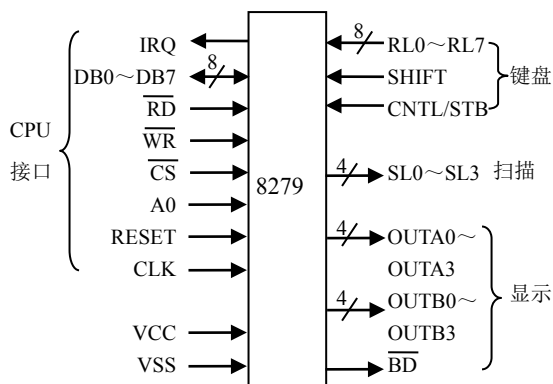
\overline{RD} 、 \overline{WD} ：读、写信号输入端，低电平有效。

IRQ：中断请求输出端，高电平有效。

SL0~SL3：扫描输出端，用于扫描键盘和显示器。可编程设定为编码（4 中选 1）或译码输出（16 选 1）。



(a) 8279 引脚排列



(b) 8279 功能方框图

图 4.17 8279 引脚排列及功能方框图

RL0~RL7：回复线，它们是键盘或传感器的列信号输入端。

SHIFT：移位信号输入端，高电平有效。它是 8279 键盘数据的次高位（D6），通常用作键盘上、下挡功能键。在传感器和选通方式中，SHIFT 无效。

CNTL/STB：控制/选通输入端，高电平有效。在键盘工作方式时，它是键盘数据的最高位，通常用作控制键。在选通输入方式时，它的上升沿可把来自 RL0~RL7 的数据存入

FIFO/传感器 RAM 中。在传感器方式时，它无效。

OUTA0~OUTA3: A 组显示信号输出端。

OUTB0~OUTB3: B 组显示信号输出端。

$\overline{\text{BD}}$: 显示熄灭输出端，低电平有效。它在数字切换显示或使用熄灭命令时关显示。

(2) 8279 的内部结构

8279 的内部结构方框图如图 4.18 所示。

DB0~DB7 是数据线，与 CPU 总线相连。当片选线 $\overline{\text{CS}}=0$ 时，选中该片。数据选择线 A0=1，数据线上信息是命令或状态；若 A0=0，数据线上信息是显示数据或键盘数据。即 A0=1、写信号线 $\overline{\text{WR}}=0$ 时，命令写到定时与控制寄存器中，对 8279 进行编程；A0=1、读信号线 $\overline{\text{RD}}=0$ 时，读 FIFO 状态寄存器的内容；A0=0、写信号线 $\overline{\text{WR}}=0$ 时，数据写到显示 RAM；当 A0=0、写信号线 $\overline{\text{RD}}=0$ 时，读 RAM 或 FIFO/传感器 RAM 的内容。

扫描计数器通过 SL0~SL3 输出扫描信号，扫描信号分别为译码和编码两种（由编程确定）。显示寄存器通过 OUTA 和 OUTB 同步输出显示 RAM 的内容。这一过程有硬件自动完成，无须程序干预。

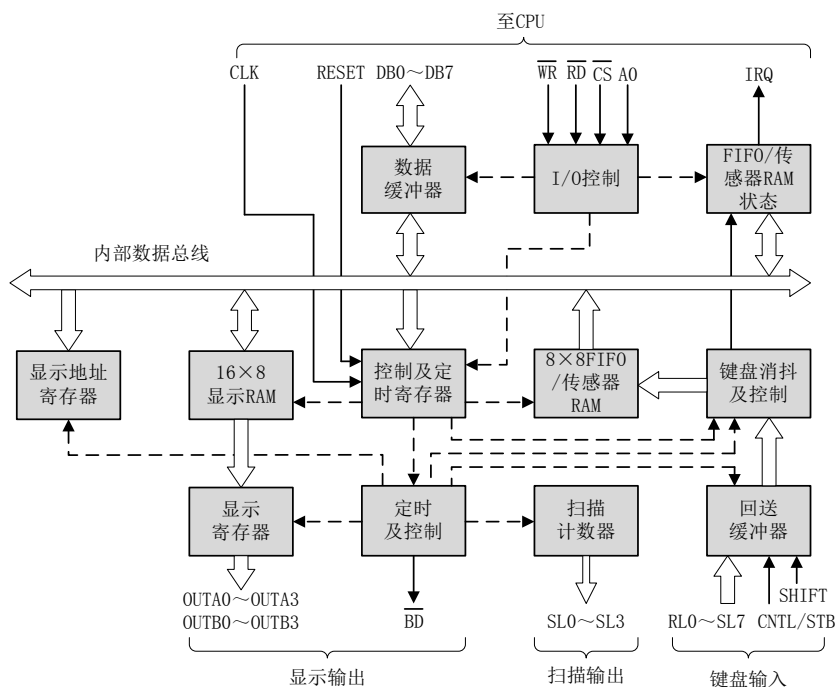


图 4.18 8279 内部结构方框图

扫描输出和回复线可以构成对键的一个扫描阵列。当有键按下时，该键在行列中的位置加上 SHIFT 和 CNTL 的状态一起被送到 FIFO 存储器中，同时使 IRQ 变高。FIFO/传感器 RAM 是一个双重功能的 8x8RAM，在键盘和选通工作方式时，它是 FIFO 存储器，其输入或输出遵循先入先出的原则。此时 FIFO 状态寄存器存放 FIFO 存储器空、满、溢出等状态。当 FIFO 存储器有数据时，IRQ 信号变为高电平，向 CPU 发出中断申请。在传感器矩阵方式工作时，FIFO/传感器 RAM 作为是传感器 RAM，它存放着传感器矩阵中的每一

个传感器的状态，若检索出传感器的变化，IRQ 信号变为高电平，向 CPU 发出中断申请。

来自 RL0~RL7 的 8 个回馈信号由回馈缓冲器加以缓冲并锁存。在键盘工作方式中，回复线作为行列式键盘的列输入线，相应的列输入信号称为回复信号，由回复缓冲器缓冲并锁存。在逐行列扫描时，回复线用来搜寻每一行列中闭合的键，当某一键闭合时，去抖电路被置位，延时等待 10ms 后，再检查该键是否仍处在闭合状态，如不是闭合，则当作干扰信号不予理睬；如是闭合，则将该键的地址和附加的移位、控制状态一起形成键盘数据被送入 8279 内部的 FIFO（先进先出）存储器。

在传感器开关状态矩阵方式中，回复线的内容直接被送往相应的传感器 RAM（即 FIFO 存储器）。在选通输入方式工作时，回复线的内容在 CNTL/STB 线的脉冲上升沿被送入 FIFO 存储器。

（3）8279 的工作方式

8279 工作方式的确定是通过 CPU 对 8279 送入命令字实现，当数据选择端 A0 置“1”时，CPU 对 8279 写入的数据为命令字，读出的数据为状态字。在叙述命令字、状态字前需先说明 8279 的 3 种工作方式。

① 键盘的工作方式。

通过对键盘/显示方式命令字的设置，可置为双键互锁方式和 N 键巡回方式。

双键互锁。

双键锁定是为两键同时按下提供的保护方法。若有两键或多个键同时按下，则无论这些键是以什么次序按下的，它只识别最后一个释放的键，并把该键值送入 FIFO/传感器 RAM 中。

N 键巡回。

N 键巡回是为 N 个键同时按下时提供的保护方法。若有多个键同时按下时，键盘扫描能按按键先后顺序依次将键值送入 FIFO/传感器 RAM 中。

② 显示器工作方式。

通过对键盘/显示方式命令字和写显示 RAM 命令字的设置，显示数据写入显示缓冲器时可置为左端送入和右端送入两种方式。左端送入为依次填入方式，右端送入为移位方式。

③ 传感器矩阵方式。

通过对读 FIFO/传感器 RAM 命令字的设置可将 8279 设置成传感器矩阵工作方式，此时，传感器的开关状态直接送到传感器 RAM。CPU 对传感器阵列扫描时，如果检测到某个传感器状态发生变化时，则中断申请信号 IRQ 变为高电平。如果 AI=0，则对传感器 RAM 的第一次读操作即清除 IRQ；如果 AI=1，则由中断结束命令清除 IRQ。

（4）8279 的命令格式和命令字

8279 共有 8 条命令，其格式及功能如下所述。

① 键盘/显示方式设置命令字。

其命令格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	D	D	K	K	K

其中:

D7、D6、D5 为 000, 是方式设置命令特征位。

DD (D4、D3): 设定显示方式, 其定义如下:

00: 8 个字符显示, 左边输入

01: 16 个字符显示, 左边输入

10: 8 个字符显示, 右边输入

11: 16 个字符显示, 右边输入

KKK (D2、D1、D0): 可设定 7 种键盘、显示工作方式, 其定义如下:

000: 编码扫描键盘, 双键锁定

001: 译码扫描键盘, 双键锁定

010: 编码扫描键盘, N 键轮回

011: 译码扫描键盘, N 键轮回

100: 编码扫描传感器矩阵

101: 译码扫描传感器矩阵

110: 选通输入, 编码显示扫描

111: 选通输入, 译码显示扫描

② 时钟编程命令。

其命令格式如下:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	P	P	P	P	P

其中:

D7、D6、D5 为 001, 是时钟编程命令特征位。

PPPPP (D4、D3、D2、D1、D0): 设定对 CLK 输入端输入的外部时钟信号进行分频的分频数 N, 用以产生 100KHz 的内部时钟, N 的取值为 2~31。若 CLK 输入的时钟频率为 2MHz, 则 N=20, PPPPP=10100B。

③ 读 FIFO/传感器 RAM 命令。

其命令格式如下:

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	AI	×	A	A	A

其中:

D7、D6、D5 为 001, 是读 FIFO/传感器 RAM 命令特征位。

AI (D4) 为自动加 1 标志。

AAA 为 FIFO/传感器 RAM 地址。

键扫描方式时, 读取数据按先进先出的原则读出, 与 AI、AAA 无关, D0~D4 可为任意值, 此时, 该命令字可设为 40H。在传感器或选通输入方式时, AAA 为 RAM 地址。当

AI=0 时，每次读完传感器 RAM 的数据后地址不变；当 AI=1 时，每次读完传感器 RAM 的数据后地址自动加 1。这样，下一个数据便从下一个地址读出，不必重新设置读 FIFO/传感器 RAM 命令。

④ 读显示 RAM 命令。

其命令格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	AI	A	A	A	A

其中：

D7、D6、D5 为 011，是读显示 RAM 命令特征位。

AI (D4) 为自动加 1 标志，AI=1 时，每次读数后地址自动加 1。

AAAA (D3、D2、D1、D0) 为显示 RAM 中的存储单元地址。

⑤ 写显示 RAM 命令。

其命令格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	AI	A	A	A	A

其中：

D7、D6、D5 为 100，是写显示 RAM 命令特征位。

AI (D4) 为自动加 1 标志，AI=1 时，每次写入数据后地址自动加 1。

AAAA (D3、D2、D1、D0) 为将要写入的显示 RAM 中的存储单元地址。

CPU 将显示数据写入显示 RAM 还必须先设置键盘/显示方式设置命令字，若选择 8 个显示器并从左端输入，键盘设为双键锁定的编码键盘方式，则应设置键盘/显示方式设置命令字为 00H。如果每次写入数据后自动加 1，且从 0 地址开始写入，则应设置写显示 RAM 命令为 90H；如要输入 10 个字符，则其输入过程如表 4.12 所示（依次填入方式）。

表 4.12 左端送入的送数过程

RAM 写入次数	AD0	AD1	AD2	AD3	AD4	AD5	AD6	AD7
第 1 次	1							
第 2 次	1	2						
...								
第 8 次	1	2	3	4	5	6	7	8
第 9 次	9	2	3	4	5	6	7	8
第 10 次	9	10	3	4	5	6	7	8

如将上述键盘/显示方式设置命令字设置为 10H，则可实现从右端输入。其输入过程如表 4.13 所示（移位方式）。

表 4.13 右端送入的送数过程

RAM 写入次数	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
第 1 次								1
第 2 次	1	2					1	2
...								
第 8 次	1	2	3	4	5	6	7	8
第 9 次	2	3	4	5	6	7	8	9
第 10 次	3	4	5	6	7	8	9	10

⑥ 显示禁止写入/消隐命令。

其命令格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	×	IWA	IWB	BLA	BLB

其中：

D7、D6、D5 为 101，是显示禁止写入/熄灭命令的特征位。

IWA、IWB（D3、D2）为 A、B 组显示 RAM 写入屏蔽位。因为显示寄存器分成 A、B 两组，可以单独送数，所以，用两位分别屏蔽。当 IWA=1 时，A 组显示 RAM 禁止写入，此时，从 CPU 写入显示器 RAM 数据不影响 A 组显示器的显示，这种情况通常用于双 4 位显示器。IWB 的用法与 IWA 相同，可屏蔽 B 组显示器。

BLA、BLB（D1、D0）为 A、B 组的消隐设置位。BLA（或 BLB）=1 则对应组的显示输出熄灭；若 BLA（或 BLB）=0 则恢复显示。

⑦ 清除命令。

其命令格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	CD	CD	CD	CF	CA

其中：

D7、D6、D5 为 110，是清除命令的特征位。

CDCDCD（D4、D3、D2）用来设定清除显示 RAM 方式，具体设置如表 4.14 所示。

表 4.14 CD 位定义的清除

CD (D4)	CD (D3)	CD (D2)	清除方式
1	0	×	将显示 RAM 全部清 0-
	1	0	将显示 RAM 置为 20H（A 组=0010，B 组=0000）
	1	1	将显示 RAM 全部置为 1
0	不清除（若 CA=1，D3、D2 仍有效）		

CF (D1) 用于清除 FIFO 存储器, CF=1 清除 FIFO 状态, 并使中断输出线复位; 同时, 传感器 RAM 的读出地址也被置 0。

CA (D0) 为总清除特征位, 兼有 CD 和 CF 的联合功能。CF=1 时, 清除显示器和 FIFO 的状态。

⑧ 结束中断/出错方式设置命令。

其命令格式如下:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	E	×	×	×	×

其中:

D7、D6、D5 为 111, 是结束中断/出错方式设置命令的特征位。

E (D4) 为 1 时, N 键轮回工作方式可工作在特殊出错方式 (多个键同时按下); 对传感器工作方式, 此命令使 IRQ 变低结束中断, 并允许对 RAM 进一步写入。

(5) 8279 状态格式与状态字

8279 的 FIFO 状态字, 主要用于键盘和选通工作方式, 以指示数据缓冲器 FIFO/传感器 RAM 中的字符数和有错误发生, 状态字节的读出地址和命令输入地址相同 ($\overline{CS}=0$, A0=1)。状态字节格式如下:

D7	D6	D5	D4	D3	D2	D1	D0
DU	S/E	O	U	F	N	N	N

其中:

DU (D7) 为显示无效特征位, DU=1 表示显示无效。显示 RAM 在清除显示或全清命令尚未完成时, DU=1, 此时对显示 RAM 操作无效。

S/E (D6) 为传感器信号结束/错误特征位, 在读 FIFO 状态字时被读出, 在执行 CF=1 时被复位。在传感器方式时, S/E=1 表示至少有一个键闭合; 在特殊出错方式时, S/E=1 表示有多键同时按下。

O (D5) 为 FIFO/传感器 RAM 溢出标志位, 当 FIFO/传感器 RAM 填满时再送入数据则该位置 1。

U (D4) 为 FIFO/传感器 RAM 空标志位, 当 FIFO/传感器 RAM 中无数据时, 如 CPU 读 FIFO/传感器 RAM 则该位置 1。

F (D3) 为 FIFO/传感器 RAM 满标志位, F=1 表示 FIFO/传感器 RAM 中已满。

NNN (D2、D1、D0) 表示 FIFO/传感器 RAM 中的字符个数, 即数据个数。

(6) 8279 的数据输入/输出

对 8279 输入数据 (如显示数据、键输入数据、传感器矩阵数据等) 时, 要选择数据输入输出地址。8279 的数据输入/输出地址由 $\overline{CS}=0$ 、A=0 确定。

在键盘扫描方式中，8279 中键输入数据按下列格式存放：

D7	D6	D5	D4	D3	D2	D1	D0
CNTL	SHIFT	行 号			列 号		

其中，CNTL（D7）为控制键 CNTL 的状态位。CNTL 为单独按键，可与其他键连用构成特殊命令。

SHIFT（D6）为控制键 SHIFT 的状态位。SHIFT 为单独按键，用做按键上、下挡控制。

行号（D5、D4、D3）为按下键所在的行号，由 RL0~RL7 的状态确定。

列号（D2、D1、D0）为按下键所在的列号，由 SL0~SL2 的状态确定。

（7）8279 的内部译码与外部译码

在键盘、显示器工作方式中 SL0~SL3 为键盘的列扫描线和动态显示的位选线。

当选择内部译码（键盘显示方式设置命令字的 D0=1）时，SL0~SL3 每一时刻只有一位为低电平输出，此时，8279 只能外接 4 位显示器和 4×8 键盘。

当选择外部译码（键盘显示方式设置命令字的 D0=0）时，SL0~SL3 成计数分频式波形输出，此时，若外接 4~16 译码器，则译码器的 16 个输出可作为外接 16 位显示器的位信号；若外接 3~8 译码器，则译码器的 8 个输出与 RL0~RL7 配合可构成 8×8 键盘（键输入数据格式中只能计入 SL0~SL2 的 8 中状态）。

（8）8279 与单片机、键盘/显示器的接口

8279 是一种功能较强的键盘/显示接口电路，可直接与 Intel 公司的各个系列的单片机接口，可以外接多种规格的键盘和显示器。由 SL0~SL2 译出键扫描线，由 4~16 译码器对 SL0~SL3 译出显示器的位扫描线。在实际应用中，键盘的大小和显示器的位数可以根据具体需要而定。

5. 用 MCS-51 的串行口扩展并行口

（1）用 74LS165 扩展并行输入口

用 74LS165 扩展并行输入口如图 4.19 所示。

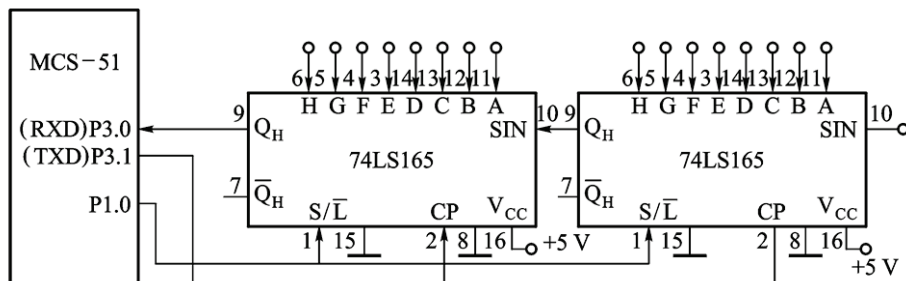


图 4.19 用 74LS165 扩展并行输入口

（2）用 74LS164 扩展并行输出口

用 74LS164 扩展并行输出口如图 4.20 所示。

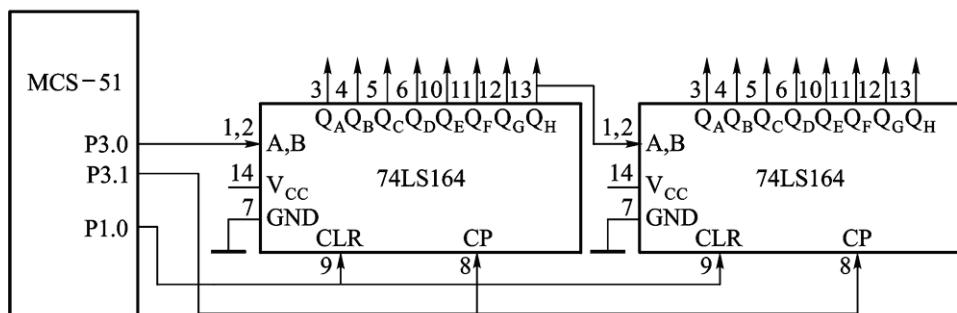


图 4.20 用 74LS164 扩展并行输出口

4.2 学习实例

实例一 用 62256 扩展 32KB 的外部 RAM

1. 实例说明

要求用 SRAM 62256 扩展一个 32KB 的 RAM，并从 5000H 单元开始写入 5 个字节的数据。仿真时单片机 P0 口要接地址锁存器 74LS373 或 74LS573 锁存器。

2. 仿真电路

用 62256 扩展 32KB 的外部 RAM 仿真电路如图 4.21 所示。

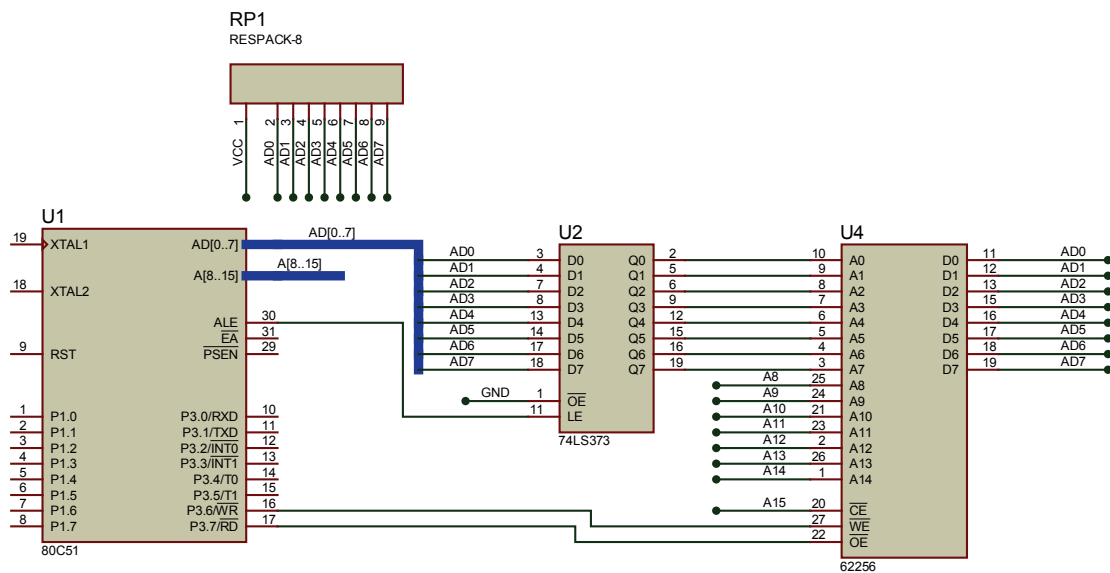


图 4.21 用 62256 扩展 32KB 的外部 RAM 仿真电路

实例三 用 AT24C02 扩展 EEPROM

1. 实例说明

要求用 AT24C02 扩展一个 256X8 的 EEPROM，将共阴极七段显示器 0~9 的段码先写入 AT24C02，然后从 P1 口依次输出送七段显示器显示。要注意 AT24C02 是 I²C 总线接口器件。

2. 仿真电路

用 AT24C02 扩展 EEPROM 仿真电路如图 4.23 所示。

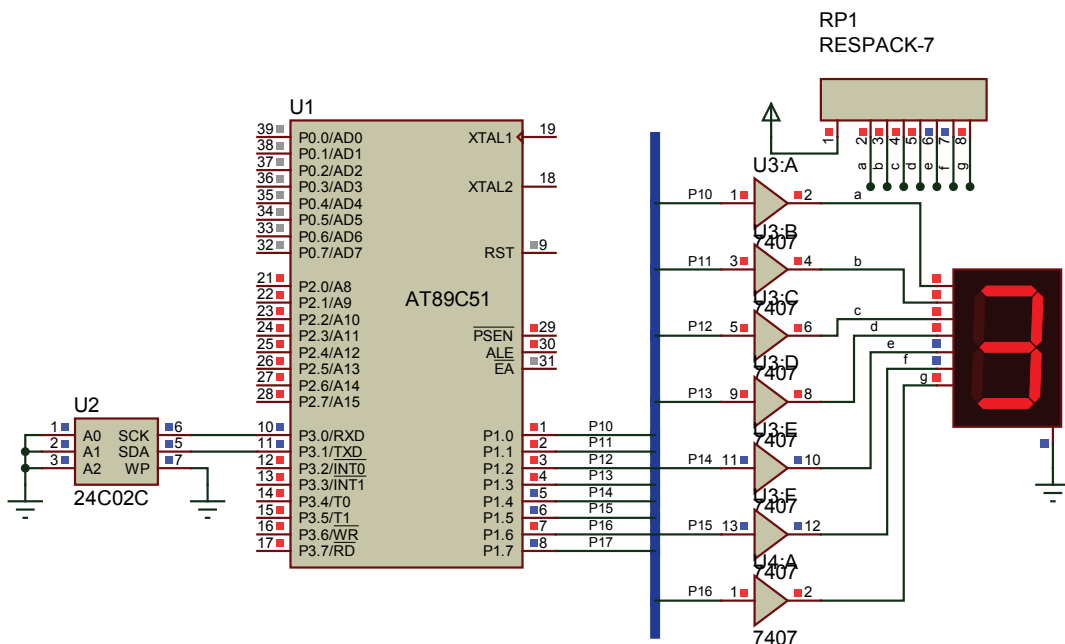


图 4.23 用 AT24C02 扩展 EEPROM 仿真电路

3. 程序设计

```
#include <reg51.h>           // 包含 51 单片机寄存器定义的头文件
#include <intrins.h>          // 包含 _nop_() 函数定义的头文件
#define READ    0xa1         // 器件地址以及读取操作, 0xa1 即为 1010 0001B
#define WRITE   0xa0         // 器件地址以及写入操作, 0xa0 即为 1010 0000B
sbit SDA=P3^1;               // 将串行数据总线 SDA 位定义在为 P3.1 引脚
sbit SCK=P3^0;               // 将串行时钟总线 SDA 位定义在为 P3.0 引脚
segcode[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
/*****
函数功能: 延时 1ms
*****/
void delay1ms()
{
```

```

    unsigned char i,j;
    for(i=0;i<10;i++)
        for(j=0;j<33;j++)
            ;
}

/*****
函数功能：延时若干毫秒
*****/
void delaynms(unsigned int n)
{
    unsigned int i;
    for(i=0;i<n;i++)
        delaylms();
}

/*****
函数功能：开始数据传送
*****/
void start()
// 开始位
{
    SDA = 1;    //SDA 初始化为高电平“1”
    SCK = 1;    //开始数据传送时，要求 SCK 为高电平“1”
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    SDA = 0;    //SDA 的下降沿被认为是开始信号
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    SCK = 0;    //SCK 为低电平时，SDA 上数据才允许变化(即允许以后的数据传递)
}

/*****
函数功能：结束数据传送
*****/
void stop()
// 停止位
{
    SDA = 0;    //SDA 初始化为低电平“0”
    SCK = 1;    //结束数据传送时，要求 SCK 为高电平“1”
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
    _nop_();    //等待一个机器周期
}

```

```

SDA = 1;    //SDA 的上升沿被认为是结束信号
_nop_();    //等待一个机器周期
_nop_();    //等待一个机器周期
_nop_();    //等待一个机器周期
_nop_();    //等待一个机器周期
SDA=0;
SCK=0;
}
/*****
函数功能：从 AT24C02 读取数据
*****/
unsigned char ReadData()
// 从 AT24C02 移入数据到 MCU
{
    unsigned char i;
    unsigned char x;    //储存从 AT24C02 中读出的数据
    for(i=0;i<8;i++)
    {
        SCK = 1;        //SCK 置为高电平
        x<<=1;          //将 x 中的各二进位向左移一位
        x|=(unsigned char)SDA; //将 SDA 上的数据通过按位“或”运算存入 x 中
        SCK = 0;        //在 SCK 的下降沿读出数据
    }
    return(x);          //将读取的数据返回
}
/*****
函数功能：向 AT24C02 的当前地址写入数据
*****/
//在调用此数据写入函数前需首先调用开始函数 start(), 所以 SCK=0
bit WriteCurrent(unsigned char y)
{
    unsigned char i;
    bit ack_bit;        //储存应答位
    for(i = 0; i < 8; i++)    // 循环移入 8 个位
    {
        SDA = (bit)(y&0x80); //通过按位“与”运算将最高位数据送到 S
        //因为传送时高位在前，低位在后
        _nop_();            //等待一个机器周期
        SCK = 1;            //在 SCK 的上升沿将数据写入 AT24C02
        _nop_();            //等待一个机器周期
        _nop_();            //等待一个机器周期
        SCK = 0;            //将 SCL 重新置为低电平，以在 SCK 线形成传送数据所需的 8 个脉冲
        y <<= 1;            //将 y 中的各二进位向左移一位
    }
    SDA = 1;                // 发送设备（主机）应在时钟脉冲的高电平期间（SCK=1）释放 SDA 线
    //以让 SDA 线转由接收设备（AT24C02）控制

```

```

    _nop_();        //等待一个机器周期
    _nop_();        //等待一个机器周期
    SCK = 1;        //根据上述规定, SCK 应为高电平
    _nop_();        //等待一个机器周期
    _nop_();        //等待一个机器周期
    _nop_();        //等待一个机器周期
    _nop_();        //等待一个机器周期
    ack_bit = SDA;  //接受设备(AT24C02)向 SDA 送低电平, 表示已经接收到一个字节
                    //若送高电平, 表示没有接收到, 传送异常
    SCK = 0;        //SCK 为低电平时, SDA 上数据才允许变化(即允许以后的数据传递)
    return ack_bit;    // 返回 AT24C02 应答位
}

/*****
函数功能: 向 AT24C02 中的指定地址写入数据
入口参数: add (储存指定的地址); dat (储存待写入的数据)
*****/
void WriteSet(unsigned char add, unsigned char dat)
// 在指定地址 addr 处写入数据 WriteCurrent
{
    start();        //开始数据传递
    WriteCurrent(WRITE); //选择要操作的 AT24C02 芯片, 并告知要对其写入数据
    WriteCurrent(add);   //写入指定地址
    WriteCurrent(dat);   //向当前地址(上面指定的地址)写入数据
    stop();           //停止数据传递
    delaynms(4);       //1 个字节的写入周期为 1ms, 最好延时 1ms 以上
}

/*****
函数功能: 从 AT24Cxx 中的当前地址读取数据
出口参数: x (储存读出的数据)
*****/
unsigned char ReadCurrent()
{
    unsigned char x;
    start();        //开始数据传递
    WriteCurrent(READ); //选择要操作的 AT24C02 芯片, 并告知要读取其数据
    x=ReadData();   //将读取的数据存入 x
    stop();         //停止数据传递
    return x;       //返回读取的数据
}

/*****
函数功能: 从 AT24C02 中的指定地址读取数据
入口参数: set_addr
出口参数: x
*****/
unsigned char ReadSet(unsigned char set_addr)
// 在指定地址读取

```



```

{
    start();                //开始数据传递
    WriteCurrent(WRITE);    //选择要操作的 AT24C02 芯片，并告知要对其写入数据
    WriteCurrent(set_addr); //写入指定地址
    return(ReadCurrent());  //从指定地址读出数据并返回
}
/*****
函数功能：主函数
*****/
void main(void)
{
    char a=0x30,b;
    SDA = 1;                // SDA=1, SCK=1, 使主从设备处于空闲状态
    SCK = 1;
    for(b=0;b<10;b++)
    {
        WriteSet(a,segcode[b]); //在指定地址“0x30”中写入数据 0
        a=a+1;
    }

    for(a=0x30;a<0x3a;a++)
    {
        P1=ReadSet(a);          //从指定地址 0x30 开始读取数据并送 P1 口显示
        delaynms(1000);
    }
}

```

实例四 用 74LS273、74LS241 扩展 I/O 接口

1. 实例说明

要求用 74LS241 扩展输入口，接八个按钮作为输入，用 74LS273 扩展输出口，接八只发光二极管，由按钮控制发光二极管点亮。设计时请查阅相关芯片手册，注意 74LS241 的 \overline{OE} 和 74LS273 的 CLK 与 OE 的译码方法。

2. 仿真电路

用 74LS273、74LS241 扩展 I/O 接口仿真电路如图 4.24 所示。

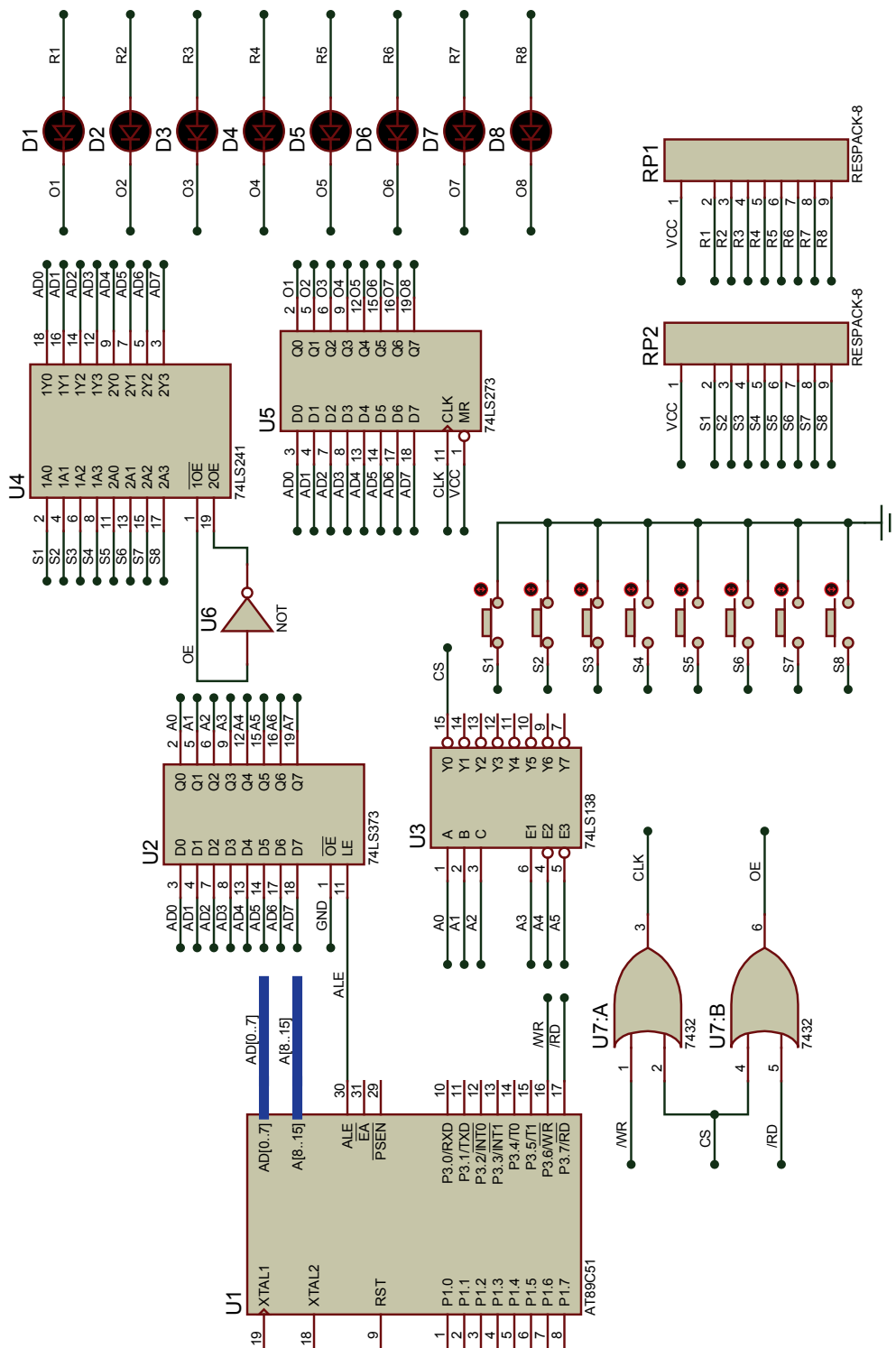


图 4.24 用 74LS273、74LS241 扩展 I/O 接口仿真电路

3. 程序设计

C 程序设计:

```
#include<reg51.h>
#include<absacc.h>
#define P273 XBYTE[0xffc8]
#define P241 XBYTE[0xffc8]
void main()
{
    while(1)
    {
        P273=P241;
    }
}
```

汇编程序设计:

```
ORG 0000H
    LJMP MAIN
ORG 1000H
MAIN:MOV DPTR,#0FFC8H
    MOVX A,@DPTR
    MOVX @DPTR,A
    LJMP MAIN
```

实例五 用 8255 芯片扩展键盘/显示接口

1. 实例说明

用 8255 扩展并行接口,其中 8255 的 A 口为键盘接口,C 口为显示接口,由七段显示按键的值。可用线反转法识别按键,要注意去抖动问题,本例采用软件去抖。

2. 仿真电路

用 8255 芯片扩展键盘/显示接口仿真电路如图 4.25 所示。

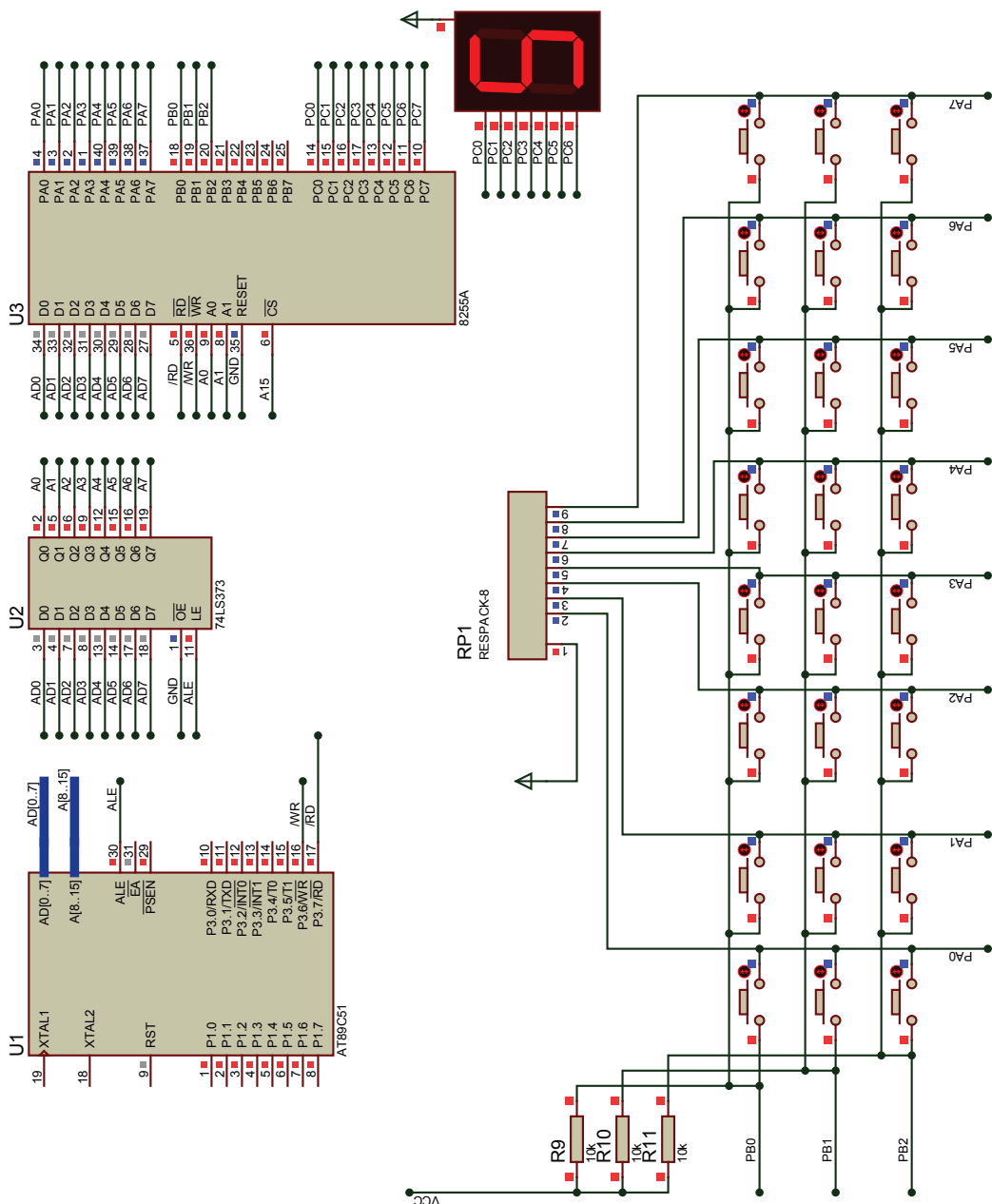


图 4.25 用 8255 芯片扩展键盘/显示接口仿真电路

3. 程序设计

```

#include <REG51.h>////////////////// MCS-51 头文件声明
#include<absacc.h>
#define uchar unsigned char////////定义无符号变量简写式
#define uint unsigned int////////
#define PA XBYTE[0x7FFC]
#define PB XBYTE[0x7FFD]
#define PC1 XBYTE[0x7FFE]
#define PCTL XBYTE[0x7FFF]
char
led[]={0xC0,0xF9,0xa4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA
1,0x86,0x8E};

void Delay(uchar); //延时程序声明（单位为毫秒）

uchar key() //键盘处理函数
{
uint a,b,c=0,d=0; //定义 4 个变量
PCTL=0x82;
PA=0x00; //键盘口置 00000000
d=PB;
d=d&0x00ff|0x00f8;
if (d!=0x00ff) //查寻键盘口的值是否变化
{
Delay(20); //延时 20 毫秒
if (d!=0x00ff) //有键按下处理
{
a=PB; //键值放入寄存器 a
a=a&0x0007;
a=a<<8;
}
PCTL=0x90;
PB=0x00; //将键盘口置为 00000000
c=PA; //将第二次取得值放入寄存器 c
c=c&0x00ff;
a=a|c; //将两个数据融合
switch(a) //对比数据值
{
case 0x06fe: b = 0x00; break; //对比得到的键值给 b 一个应用数据
case 0x06fd: b = 0x01; break;
case 0x06fb: b = 0x02; break;
case 0x06f7: b = 0x03; break;
case 0x06ef: b = 0x04; break;
case 0x06df: b = 0x05; break;
case 0x06bf: b = 0x06; break;

```

```

        case 0x067f: b = 0x07; break;
        case 0x05fe: b = 0x08; break;
        case 0x05fd: b = 0x09; break;
        case 0x05fb: b = 0x0a; break;
        case 0x05f7: b = 0x0b; break;
        case 0x05ef: b = 0x0c; break;
        case 0x05df: b = 0x0d; break;
        case 0x05bf: b = 0x0e; break;
        case 0x057f: b = 0x0f; break;
        default:    break;//键值错误处理
    }
}
return(b); //将b作为返回值
}

void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //一个 ; 表示空语句, CPU 空转
    } //i 从 0 加到 125, CPU 大概就耗时 1 毫秒
}

void main(void)
{
    while(1)
    {
        Delay(10);
        PC1=led[key()];
    }
}

```

实例六 用 8155 芯片扩展显示接口

1. 实例说明

要求用 8155 扩展并行接口, 8155 的 A 口作为输出口, 输出七段显示器的段码, B 口作为输出口, 输出七段显示器的位选码。由七段显示器显示“HELLO”。

2. 仿真电路

用 8155 芯片扩展显示接口仿真电路如图 4.26 所示。

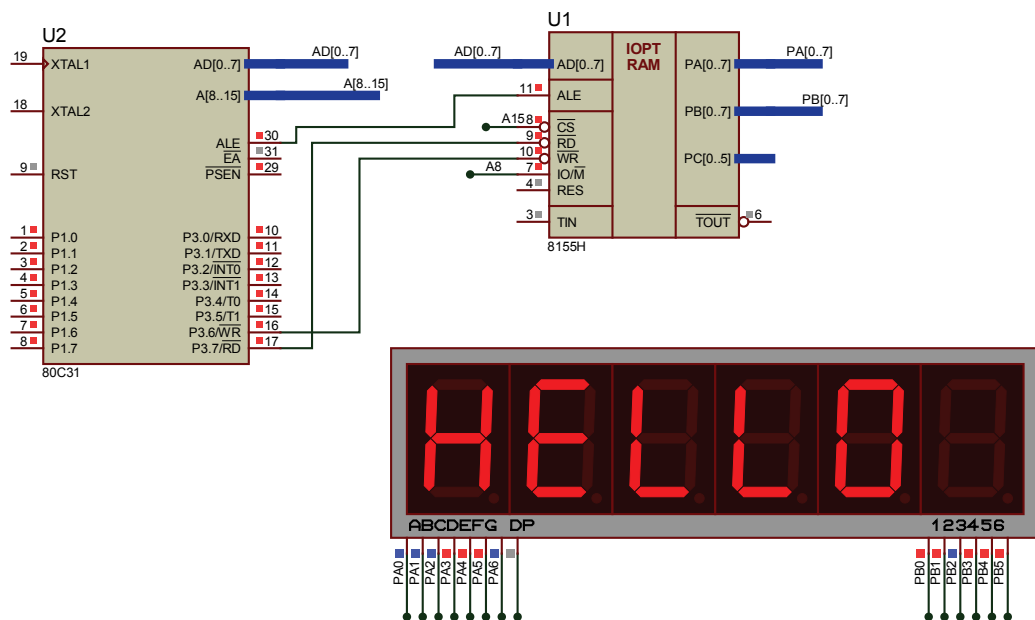


图 4.26 用 8155 芯片扩展显示接口仿真电路

3. 程序设计

```
#include<reg51.h>
#include<absacc.h> //绝对地址访问头文件
#define PA XBYTE[0x7f01] //定义 A、B、C 和控制口地址
#define PB XBYTE[0x7f02]
#define PC XBYTE[0x7f03]
#define PK XBYTE[0x7f00]
unsigned char dispcode[]={0x76,0x79,0x38,0x38,0x3f};
unsigned char k[]={0xfe,0xfd,0xfb,0xf7,0xef};
void delay100(int n)
{
    while(n--)
    {
        int i,j;
        for(i=0;i<10;i++)
            for(j=0;j<10;j++)
                ;
    }
}
void main()
{
    PK=0x03;
    while(1)
    { int i;
```

```

for(i=0;i<5;i++)
{
    PA=dispcode[i];
    PB=k[i];
    delay100(5);
    PB=0xff;
}
}
}

```

实验七 用 74ls165、74ls164 扩展键盘/显示接口

1. 实例说明

要求用单片机的串行口工作在方式 0，用 74ls165 扩展键盘接口，74ls164 扩展输出接口。

2. 仿真电路

用 74ls165、74ls164 扩展键盘/显示接口仿真电路如图 4.27 所示。

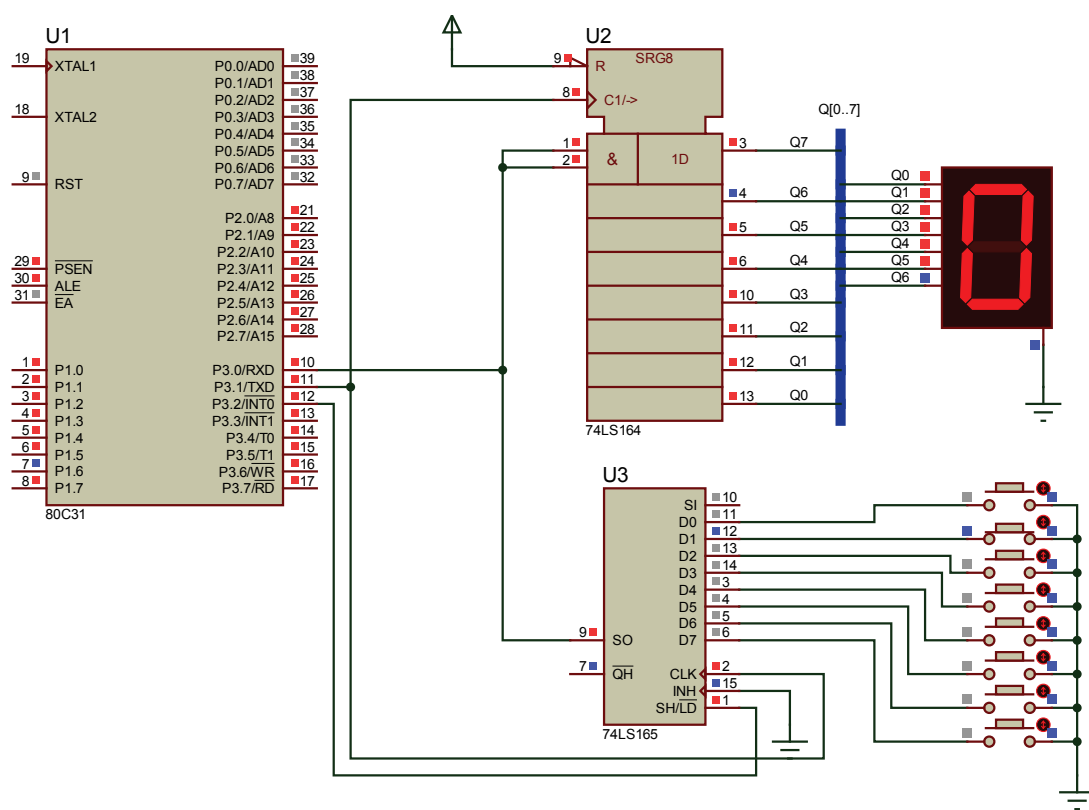


图 4.27 用 74ls165、74ls164 扩展键盘/显示接口仿真电路

3. 程序设计

```
#include <reg51.h>
#include <intrins.h>
#define uint unsigned int
#define uchar unsigned char
sbit SHL=P3^2;
void delay(uint x)
{
    uchar i;
    while(x--)
        for(i=0;i<120;i++);
}

void main()
{
    SCON=0X10; //串行口工作在方式 0, REN=1, 允许串口接收
    while(1)
    {
        unsigned char a,b=0;
        SHL=0; //置数, 读入并行输入口的 8 位数据
        SHL=1; //移位, 串口输入被封锁, 串行转换开始
        while(RI==0); //等待接收完一字节数据
        RI=0; //接收完成, RI 置 0
        a=SBUF; //接收到的字节送 a
        P1=a;
        delay(20);
        SBUF=a;
        while(TI==0);
        TI=0;
    }
}
```

本章小结

本章主要介绍了单片机的外围扩展数字电路芯片的工作原理、系统总线结构及存储器和 I/O 接口的扩展方法。通过实例让读者对单片机外围接口扩展技术及应用有了比较正确的认识, 单片机的应用范围越来越大, 接口电路也不断的发展创新, 只有掌握了常用接口的扩展方法, 才能在实际应用中更好地使用这些接口。

习题四

一、填空题

1. 74LS273 通常用来作简单_____ (输出/输入) 接口扩展; 而 74LS244 则常用来做简单_____ (输出/输入) 接口扩展。

2. 8031 外部程序存储器的最大可扩展容量是_____, 其地址范围是_____。
ROM 芯片 2764 的容量是_____, 若其首地址为 0000H, 则其末地址_____。
3. 通过 CPU 对 I/O 状态的测试, 只有 I/O 已准备好时才能进行 I/O 传送, 这种传送方式称为_____。
4. 半导体存储器分成两大类_____和_____, 其中_____具有易失性, 常用于存储_____。
5. 8255A 属于可编程的_____ I/O 接口芯片, 8255A 的 A 通道有_____种工作方式。

二、单项选择题

1. MCS-51 外扩 ROM, RAM 和 I/O 口时, 它的数据总线是 ()。
A. P0 口 B. P1 口 C. P2 口 D. P3 口
2. 使用 8155 可以扩展出的 I/O 口线是 ()。
A. 16 根 B. 24 根 C. 22 根 D. 32 根
3. 某种存储器芯片是 8KB×4/片, 那么它的地址线根数是 ()。
A. 11 根 B. 12 根 C. 13 根 D. 14 根
4. 6264 芯片是 ()。
A. E2PROM B. RAM
C. Flash ROM D. EPROM
5. 在 MCS-51 系统中, I/O 端口地址分配采用 ()。
A. I/O 端口地址独立编排 B. I/O 端口地址与存储器地址统一编排
C. I/O 端口地址直接寻址 D. I/O 端口全部集成于内部
6. 使用 8255 可以扩展出的 I/O 口线是 ()。
A. 16 根 B. 24 根 C. 22 根 D. 32 根
7. 不属于单片机与输入输出设备进行信息交换的方式是 ()。
A. 无条件传送方式 B. 查询方式
C. 中断方式 D. 存储器直接存取方式

三、编程设计题

1. 利用单片机的串行口和 74LS164 扩展一显示接口, 在 7 段显示器上循环显示 0~9 的数字。(1) 完成电路设计; (2) 编写程序。
2. 在单片机的外部用 8255 扩展设计一个显示接口, 动态显示 “HELLO” 字样。(1) 根据所给芯片完成电路设计。(2) 根据电路完成程序设计。

3. 有关 8255A 的使用问题。如图 4.28 所示，问：

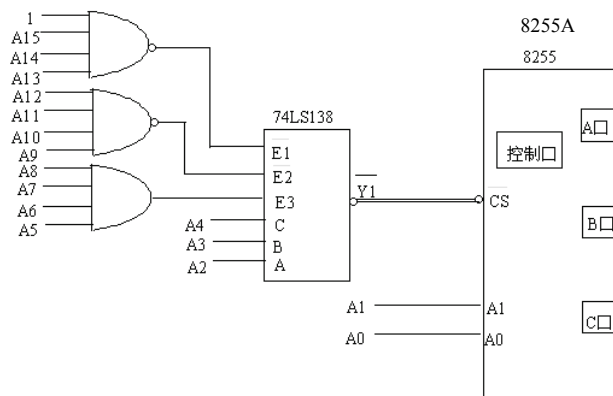


图 4.28 8255A 的使用

- (1) 图的连接中，若 CPU 有 16 条地址线，则 8255A 的控制端口地址为：_____。
- (2) 若把 8255A 的 A 口置为方式 0，输出方式；C 口置为输出方式；B 口置为方式 0 输入方式。则方式控制字为：_____。写出初始化程序段。
- (3) 若把 8255A 的 C 口第三位 PC3 置位，则控制字为：_____。

4. 用 6264 (8K*8) 构成 16KB 的数据存储系统。要求采用线选法产生片选信号，并计算 6264 的地址范围。



第5章 DAC 和 ADC 接口

学习目标

理解 ADC0809 和 DAC0832 的工作原理及引脚功能和内部结构，掌握 ADC0809 和 DAC0832 与单片机的接口方法；掌握单片机开关量接口方法。

重点难点

ADC0809 和 DAC0832 的接口设计及应用编程。

5.1 知识结构

单片机用于智能仪表和测控系统时，需要处理大量的外部信息，其中有随时间连续变化的模拟信号，也有开关信号。

工程实践中经常遇到被测对象的一些物理参数（如温度、流量、压力、位移、速度等），这些参数都是模拟量。这些模拟量由传感器、变送器变换成标准的电压或电流信号，通过模拟量/数字量（A/D）转换器，将其转换成计算机能处理的相应的数字信号。计算机对模拟设备进行控制时，如控制电动调节阀、模拟调速系统、模拟记录仪等，就需要将计算机输出的数字信号通过数字量/模拟量（D/A）转换器，转换成外设能接收的相应的模拟信号。

开关信号来自开关类器件的输入，如拨盘开关、扳键开关、继电器的触点等。当计算机输出控制的对象是具有开关状态的设备时，计算机的输出就应为开关量。一个开关只需一位二进制数（0 或 1）就可以表示其两个状态（开或关），所以 8 位字长的计算机一次就可以读入或输出 8 个开关量。

单片机与模拟信号的输入、输出通道接口技术，以及与开关量输入、输出接口技术是构成单片机测控系统的重要内容。本章主要介绍 ADC0809 和 DAC0832 的引脚功能和内部结构。

5.1.1 A/D 转换器件

1. A/D 转换器概述

能够将模拟量转换成数字量的器件称为模数转换器，简称 A/D 转换器或 ADC。数字系统所能识别并处理的都是数字量，然而在一些实际的测控系统中所发生的各种物理参数常常是一些模拟量（如温度、压力等），因此对于这样的物理量首先将其转换为电信号（电压或电流），再经 A/D 转换器转换成数字信号，传送给数字系统进行处理。

(1) A/D 转换器的分类

A/D 转换器的种类很多,就位数来分,有 8 位、10 位、12 位和 16 位等。其位数越高其分辨率就越高,价格也越贵。根据 A/D 转换器的工作原理分,常见的有计数式 A/D 转换器、双积分式 A/D 转换器、逐次逼近式 A/D 转换器、并行直接比较式 A/D 转换器、V/F 式 A/D 转换器等。其中双积分式 A/D 转换器和逐次逼近式 A/D 转换器为常用的两种,前者具有抗干扰能力强、转换精度高的优点,但速度较慢;后者速度较快,外围元件少,但其抗干扰能力较差。

(2) A/D 转换器技术指标

A/D 转换器的主要技术指标有:

分辨率: 改变一个相邻数码所需输入的模拟电压的变化量,通常用位数表示。

转换误差和精度: 指一个实际 A/D 转换器量化值与一个理想的 A/D 转换器量化值之间的最大偏差,通常以最低有效位的倍数给出。转换误差和分辨率共同描述 A/D 转换器的转换精度。

转换时间和转换速度: A/D 转换器完成一次转换所需要的时间为转换时间,每秒完成转换的快慢为转换速度。

2. ADC0808/0809 简介

(1) ADC0808/0809 的引脚功能

ADC0808/0809 是一个逐次逼近式 8 路模拟输入、8 位数字量输出的 A/D 转换器,其引脚图如图 5.1 所示。

各引脚功能如下:

IN0~IN7: 8 路模拟通道输入信号,通过模拟开关实现 8 路模拟输入信号分时选通。

A、B、C 模拟通道选择: CBA=000~111,分别选中 IN0~IN7。

ALE: 地址锁存信号,其上升沿锁存 C、B、A 的信号,译码后控制模拟开关,接通 8 路模拟输入中相应的一路。

OE、START、CLK 为控制信号端: OE 输出允许信号,START 为启动信号输入端,CLK 为时钟信号输入端。

D0~D7: 8 位数字量输出端。

EOC: 转换结束信号,启动转换后,EOC 变为低电平,转换完成后变为高电平。

VR(+)、VR(-): 参考电压输入端。

(2) ADC0808/0809 的内部结构及工作原理

ADC0808/0809 由两部分组成,如图 5.2 所示。

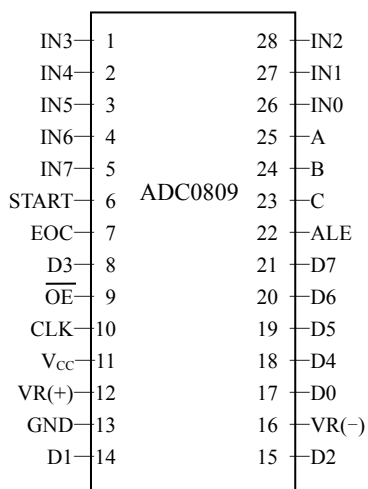


图 5.1 ADC0809 引脚图

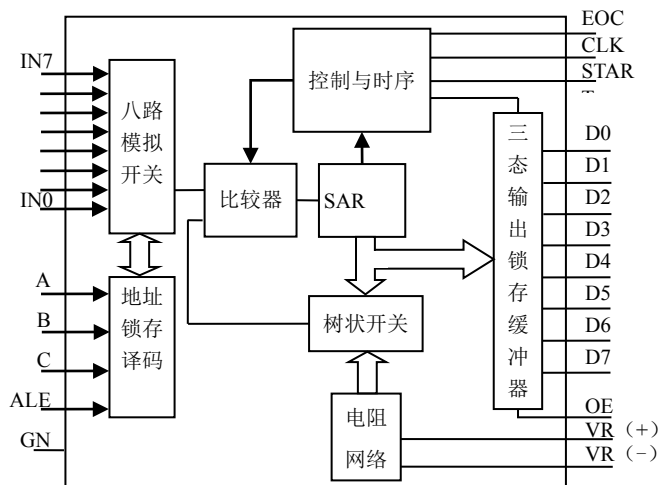


图 5.2 ADC0808/0809 的原理结构方框图

① 为 8 通道多路模拟开关以及相应的通道地址锁存与译码电路，可以实现 8 路模拟信号的分时采集，3 个地址信号 A、B 和 C 选择通道。

② 为一个逐次逼近式 A/D 转换器，由比较器、控制逻辑、三态输出缓冲器、逐次逼近寄存器以及开关树和电阻网络组成。开关树和电阻网络构成 D/A 转换器。

逻辑控制逐次逼近寄存器从高位至低位，逐次取“1”，然后将此数字量送 D/A 转换器输出一个模拟电压 V_s ， V_s 与输入模拟量 V_x 在比较器中进行比较，当 $V_s > V_x$ 时，该位 $D_i = 0$ ，若 $V_s \leq V_x$ 时，该位 $D_i = 1$ ，因此从 D7 到 D0 逐位逼近并比较 8 次，逐位逼近寄存器中的数字量，即为与模拟量 V_x 所对应的数字量。此数字量送入输出锁存器，并同时发出转换结束信号 EOC，表示一次转换结束。此时，CPU 发出一个输出允许命令 OE，即可读取数据。

3. ADC0809 与单片机接口

ADC0809 与单片机接口可以采用查询方式和中断方式。具体实例见本章实例一。

5.1.2 D/A 转换器件

1. D/A 转换器概述

D/A（数/模）转换器是一种将数字信号转换成模拟信号的器件，为计算机系统的数字信号和模拟环境的连续信号之间提供了一种接口。D/A 转换器有两种输出形式，一种是电压输出形式，即输入的是数字量，输出为电压；还有一种电流输出形式，即输出为电流。在实际应用中如需要电压模拟量的话，需将电流输出的 D/A 转换器的输出端用运算放大器组成的 I/V 转换器将电流输出转换成电压输出。模拟量转换需要一定的时间，为保持 D/A 转换器输出的稳定，就必须在微处理器与 D/A 转换器之间增加数据锁存功能，目前常用的 D/A 转换器内部都带有数据锁存器。

D/A 转换器主要的技术指标有分辨率、建立时间和转换精度。

分辨率：指输入的单位数字量变化引起的模拟量输出的变化，通常定义为最小输出电

压与最大输出电压之比，或用数字输入信号的有效位表示。

建立时间：描述 D/A 转换快慢的一个参数，其值为从输入数字量到输出模拟量稳定到相应数值范围内所经历的时间。

转换精度：用来描述转换后的实际转换特性与理想转换特性之间的最大偏差。理想情况下，精度与分辨率基本一致，位数越多精度越高。但电源电压、参考电压、电阻等各种因素存在着误差，因此精度与分辨率并不完全一致。

2. DAC0832 简介

(1) DAC0832 结构及其原理

DAC0832 采用 CMOS 工艺，具有 20 个引脚双列直插式单片 8 位 D/A 转换器，其结构方框图如图 5.3 所示。片内有两个数据缓冲器：输入寄存器和 DAC 寄存器，其控制端/LE1 和/LE2 分别受 ILE、/CS、/WR1 和/WR2、/XFER 的控制。DI0~DI7 为数据输入线，转换结果从 I_{OUT1} 、 I_{OUT2} 以模拟电流形式输出。当输入数字为全 1 时， I_{OUT1} 最大；当为全 0 时，其值最小， I_{OUT1} 和 I_{OUT2} 之和为常数。当需要输出模拟电压时，需外接运算放大器进行 I/V 转换。

DAC0832 可以构成双缓冲、单缓冲和直通工作方式。双缓冲工作方式主要用于多个 DAC0832 同步输出的情况，先分别将所有芯片的 ILE、/CS、/WR1 同时有效，将各输出数据逐个写入到各自的输入数据寄存器中。然后让所有的/WR2 和/XFER 同时为低电平，则完成同步模拟量的输出；单缓冲方式用于单一的 DAC0832 输出的情况，通常把/WR2 和/XFER 直接接地，对 DAC0832 进行写操作；直通数据输入方式由于连续数字反馈控制回路，只要把 ILE 接高电平，/CS、/WR1、/WR2 和/XFER 接地即可。

(2) DAC0832 的引脚功能

DAC0832 的引脚排列，如图 5.4 所示。

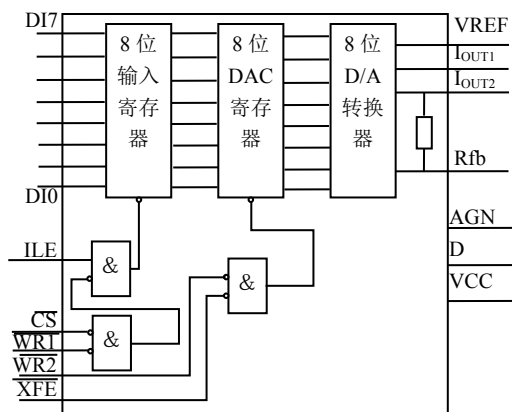


图 5.3 DAC0832 结构方框图

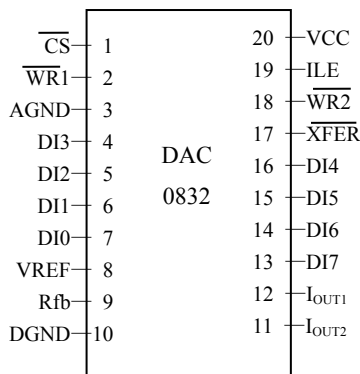


图 5.4 DAC0832 引脚排列图

DI0~DI7：8 位数字信号输入端，与单片机的数据总线相连 DI7 为最高位。

/CS：片选端，低电平有效。

ILE：数据锁存允许控制端，高电平有效。

/WR1：第一级锁存写选通，低电平有效。当/CS=0，ILE=1，/WR1=0 时，数据信号被

锁存到第一级 8 位输入寄存器中。

/XFER: 数据传送控制, 低电平有效。

/WR2: 第二级锁存写选通, 低电平有效。当/XFER=0, /WR2=0 时, 锁存器中的数据传送到 DAC 寄存器中进行转换。

I_{OUT1} : D/A 转换器电流输出 1, 输入数字量为全 1 时, I_{OUT1} 输出最大; 输入数字量为全 0 时, I_{OUT1} 输出最小。

I_{OUT2} : D/A 转换器电流输出 2, $I_{OUT1}+I_{OUT2}$ =常数。

Rfb: 外部反馈信号输入端, 为外部运算放大器提供一个反馈电压。Rfb 可由内部提供, 也可由外部提供。

VREF: 参考电压输入, 要求外部接一个精密的电源。

VCC: 数字电路供电电压, 一般为+5~+15V。

DGND: 数字信号地; AGND: 模拟信号地。

3. DAC0832 与单片机的接口

DAC0832 可工作于单缓冲、双缓冲及直通 3 种方式。

(1) 单缓冲工作方式

单缓冲方式, 即输入锁存器和 DAC 寄存器相应的控制信号引脚分别连在一起, 使数据直接写入 DAC 寄存器, 立即进行 D/A 转换 (输入锁存器不起锁存作用)。此方式适用于只有一路模拟量输出, 或有几路模拟量输出但并不要求同步的系统。具体实例见本章实例二。

(2) 双缓冲器工作方式

对于多路 D/A 转换输出, 如果要求同步进行, 就应该采用双缓冲器同步方式。DAC0832 工作于双缓冲器工作方式时, 数字量的输入锁存和 D/A 转换是分两步完成的。首先 CPU 的数据总线分时地向各路 D/A 转换器输入要转换的数字量并锁存在各自的输入锁存器中, 然后 CPU 对所有的 D/A 转换器发出控制信号, 使各个 D/A 转换器输入锁存器中的数据打入 DAC 寄存器, 实现同步转换输出。

(3) 直通工作方式

当 DAC0832 芯片的片选信号/CS、写信号/WRI、/WR2 及传送控制信号/XFER 的引脚全部接地, 允许输入锁存信号 I LE 引脚接+5V 时, DAC0832 芯片就处于直通工作方式, 数字量一旦输入, 就直接进入 DAC 寄存器, 进行 D/A 转换。

5.2 学习实例

实例一 基于 ADC0809 的 5V 直流电压表设计

1. 实例说明

要求用单片机和 ADC0808 (ADC0809) 设计一个 0~5V 的直流电压表, 测量由 IN0 输入的直流电压, 并由七段显示器显示测量的直流电压值。

2. 仿真电路

基于 ADC0809 的 5V 直流电压表设计仿真电路如图 5.5 所示。

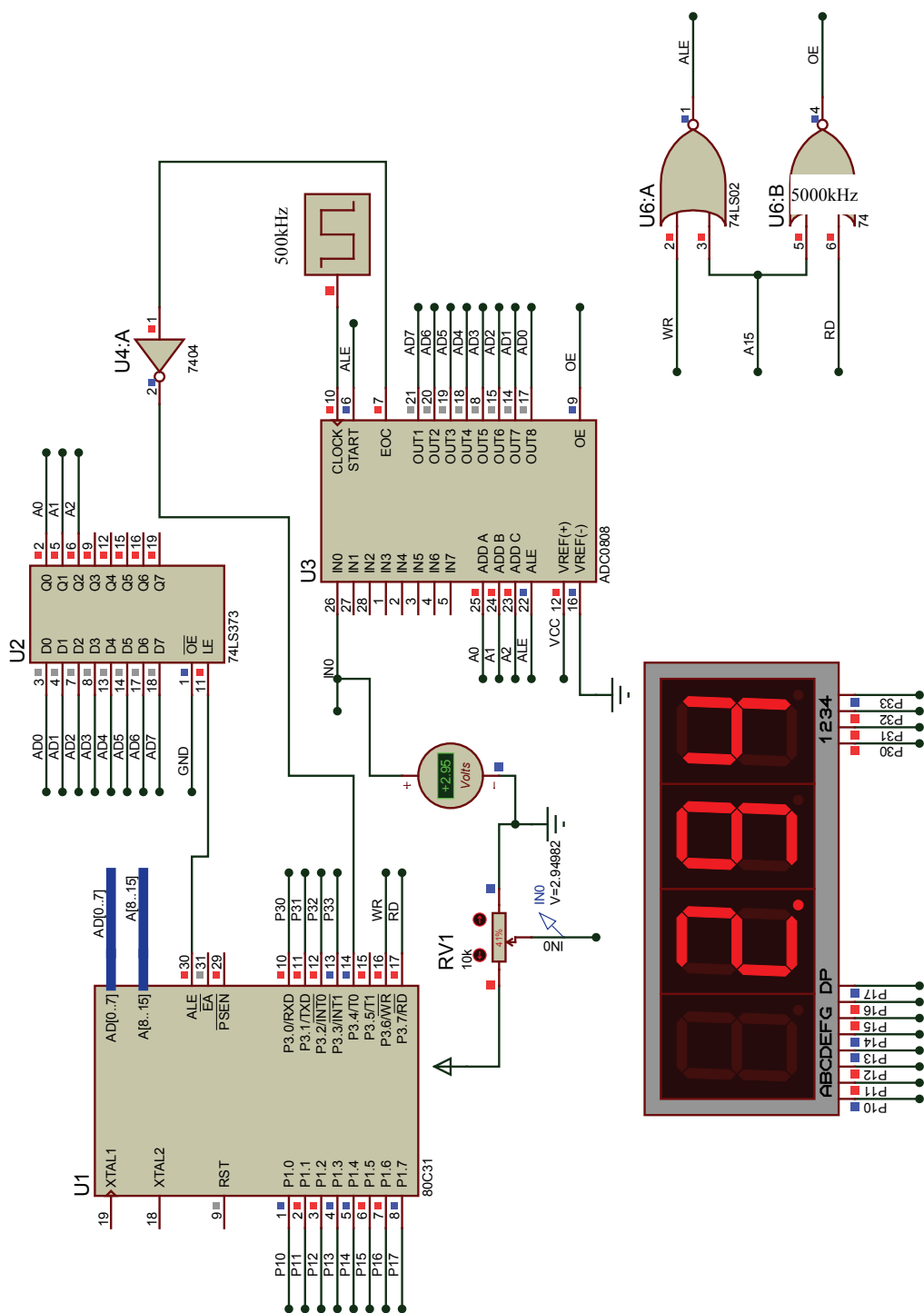


图 5.5 基于 ADC0809 的 5V 直流电压表设计仿真电路

3. 程序设计

```
#include<reg51.h>
#include<absacc.h>
#define ADC0808 XBYTE[0x7ff8] //ADC0808 地址
sbit EOC=P3^4;
char led[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F}; //段码
char ledk[]={0xfe,0xfd,0xfb,0xf7}; //位选
void delay(unsigned char j)
{
    while(j--);
}
void main(void)
{
    unsigned char a,zh,xiao;
    ADC0808=0; //启动转换
    while(1)
    {
        if(!EOC)
        {
            a=ADC0808; //读取转换结果
            zh=a/51; //计算电压值
            xiao=(a%51)*100/51;
            P3=ledk[1];
            P1=led[zh]|0x80;
            delay(1000);
            P3=ledk[2];
            P1=led[xiao/10];
            delay(1000);
            P3=ledk[3];
            P1=led[xiao%10];
            delay(1000);
            ADC0808=0;
        }
    }
}
```

实例二 用 DAC0832 设计简易信号发生器

1. 实例说明

要求用单片机和 DAC0832 设计一个简易信号发生器，能产生方波、三角波、锯齿波和正弦波等波形。

2. 仿真电路

用 DAC0832 设计简易信号发生器仿真电路如图 5.6 所示。

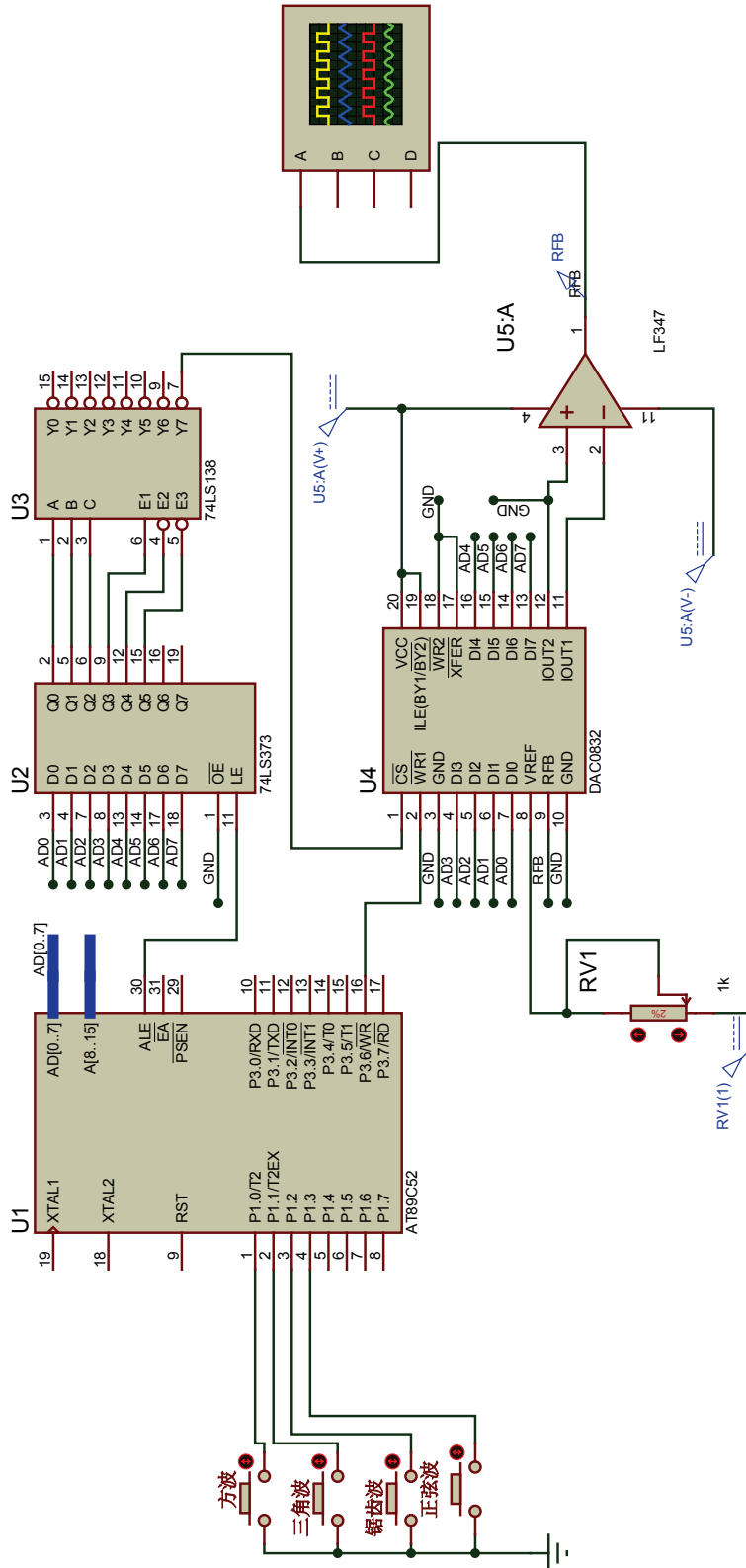


图 5.6 用 DAC0832 设计简易信号发生器仿真电路

3. 程序设计

```

#include<reg51.h>
#include<math.h>
#include<absacc.h>
#define DAC0832 XBYTE[0xffcf] //DAC0832 地址
code unsigned char Sin[128]={64,67,70,73,76,79,82,85,88,91,94,96,99,102,
104,106,109,111,113,115,117,118,120,121,123,124,125,126,126,127,127,127,127,
127,127,127,126,126,125,124,123,121,120,118,117,115,113,111,109,106,104,102,
99,96,94,91,88,85,82,79,76,73,70,67,64,60,57,54,51,48,45,42,39,36,33,31,28,
25,23,21,18,16,14,12,10,9,7,6,4,3,2,1,1,0,0,0,0,0,0,0,1,1,2,3,4,6,7,9,10,12,
14,16,18,21,23,25,28,31,33,36,39,42,45,48,51,54,57,60}; //正弦描点
void delay(unsigned int j)
{
    while(j--);
}
void fangbo(void) //方波
{
    DAC0832=0xff;
    delay(500);
    DAC0832=0x00;
    delay(500);
}

void sanjiao(void) //三角波
{
    unsigned char i;
    while(1)
    {
        for(i=0;i<255;i++)
            DAC0832=i;
        for(i=255;i>0;i--)
            DAC0832=i;
        DAC0832=0;
        {if(P1!=0xfd)
            break;}
    }
}

void juchi(void) //锯齿波
{
    unsigned char i;
    while(1)
    {
        for(i=0;i<=255;i++)
        {
            DAC0832=i;
            if(P1!=0xfb)
                break;
        }
    }
}

```

```

    }
    }
}

void zhengxuan(void) //正弦波
{
    unsigned char i;
    while(1)
    {
        for(i=0;i<128;i++)
        { DAC0832=Sin[i];
          if(P1!=0xf7)
            break;}
        }
    }

void main(void)
{
    unsigned char a;
    P1=0xff;
    a=P1;
    switch(a) //波形选择
    {
        case 0xfe:fangbo();break;
        case 0xfd:sanjiao();break;
        case 0xfb:juchi();break;
        case 0xf7:zhengxuan();break;
        default:break;
    }
}

```

本章小结

本章主要介绍了 D/A、A/D 转换器的原理及应用。通过两个实例让读者对 D/A 和 A/D 转换器有初步的了解。

习题五

一、填空题

1. A/D 转换器的三个最重要指标是：_____、_____和_____。
2. 对于 D/A 转换器内部没有带寄存器的芯片，当微处理器与其传送信息时，必须通过_____或_____交换信息。
3. 传感器是能把_____的模拟量转换成电流或_____信号。

4. 一般模拟信号变化_____, 可以用_____形成通路来监视或控制。
5. 模拟信号是连续不断变化的, 而 A/D 转换总需要一定的_____, 所以需要把待转换的信号采样后还_____一段时间。
6. 采样保持电路广泛用于_____系统和_____系统之中。
7. A/D 转换需要一段时间, 所以输入信号变化速率_____时, 应采用_____电路。
8. 为减少量化误差, 可以采用_____更多的 A/D 转换器, 把模拟范围分割成_____离散区间。

二、单项选择题

1. 要想把数字送入 DAC0832 的输入缓冲器, 其控制信号应满足 ()。
 - A. $\text{ILE}=1, \overline{\text{CS}}=1, \overline{\text{WR}}_1=0$
 - B. $\text{ILE}=1, \overline{\text{CS}}=0, \overline{\text{WR}}_1=0$
 - C. $\text{ILE}=0, \overline{\text{CS}}=1, \overline{\text{WR}}_1=0$
 - D. $\text{ILE}=0, \overline{\text{CS}}=0, \overline{\text{WR}}_1=0$
2. A/D 转换方法有以下四种, ADC0809 是一种采用 () 进行 A/D 转换的 8 位接口芯片。
 - A. 计数式
 - B. 双积分式
 - C. 逐次逼近式
 - D. 并行式

三、编程设计题

1. 设 DAC0832 接口电路如图 5.7 所示, 试编写实现输出三角波形的程序 (0832 地址为 FEH)。

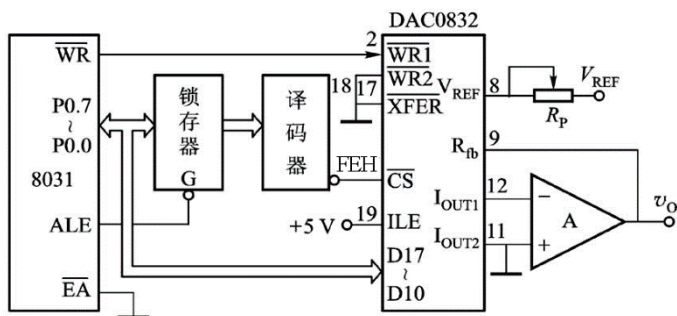


图 5.7 DAC0832 接口电路

2. 图 5.8 所示是 DAC0832 的应用电路, DA 转换时数字量 FFH 与 00H 分别对应于模拟量 +5V 与 0V。图右下方给出了 DAC0832 的逻辑结构。
 - (1) 将图中空缺的电路补充完整;
 - (2) 编写程序, 产生图中所示锯齿波。设有一个延时 3.905ms 的子程序 DELAY 可以直接调用。

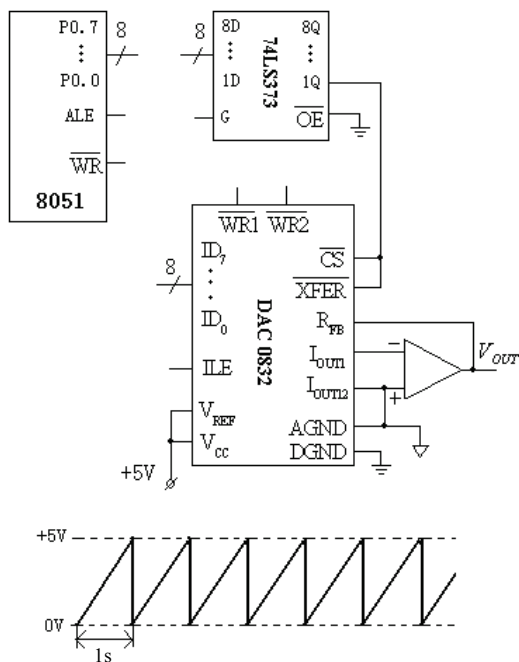


图 5.8 DAC0832 的应用电路

3. 转换 ADC0809 的 IN7 输入的 0~5V 电压，并把转换结果通过 7 段显示器显示出来。请设计硬件电路并编程实现。

第6章 键盘与显示接口设计

学习目标

掌握单片机的键盘、显示接口设计方法。

重点难点

- (1) 键盘的扫描原理及扫描程序和编写。
- (2) LCD 初始化程序的编写。

6.1 知识结构

6.1.1 键盘接口设计

键盘是单片机应用系统中最常用的输入设备，操作人员一般都是通过键盘向单片机系统输入指令、数据，实现简单的人机通信。

6.1.1.1 键盘接口概述

键盘可分为编码键盘与非编码键盘两类，编码键盘主要是用硬件来实现对键的识别，非编码键盘主要是由软件来实现键盘的定义与识别。

全编码键盘能够由硬件逻辑自动提供与键对应的编码，此外，一般还具有去抖动和多键、串键保护电路。这种键盘使用方便，但需要较多的硬件，价格较贵，一般的单片机应用系统较少采用。非编码键盘只简单地提供行和列的矩阵，其他工作均由软件完成。由于其经济实用，较多地应用于单片机系统中。下面将重点介绍非编码键盘接口。

(1) 按键结构与特点

键盘通常使用机械触点式按键开关，其主要功能是把机械上的通断转换成为电气上的逻辑关系。也就是说，它能提供标准的 TTL 逻辑电平，以便与通用数字系统的逻辑电平相容。机械式按键再按下或释放时，由于机械弹性作用的影响，通常伴随有一定时间的触点机械抖动，然后其触点才稳定下来。其抖动过程如图 6.1 所示，抖动时间的长短与开关的机械特性有关，一般为 5~10 ms。

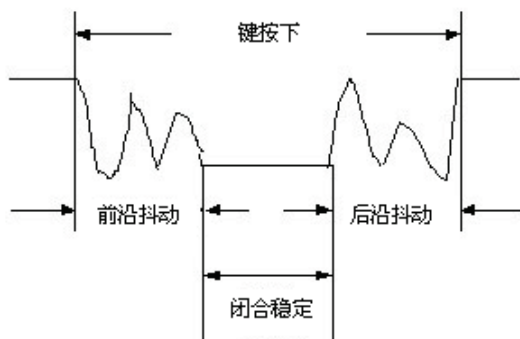


图 6.1 按键开关产生的波形

(2) 按键抖动的消除

在触点抖动期间检测按键的通与断状态，可能导致判断出错，即按键一次按下或释放被错误地认为是多次操作，这种情况是不允许出现的。为了克服按键触点机械抖动所致的检测误判，必须采取去抖动措施。可采用硬件和软件方法去抖动。通常在键数较少时，用硬件去抖，键数较多时，采用软件去抖。

在硬件上可采用在键输出端加 R-S 触发器（双稳态触发器）或单稳态触发器构成去抖动电路。图 6.2 所示是一种由 R-S 触发器构成的去抖动电路，当触发器一旦翻转，触点抖动不会对其产生任何影响。

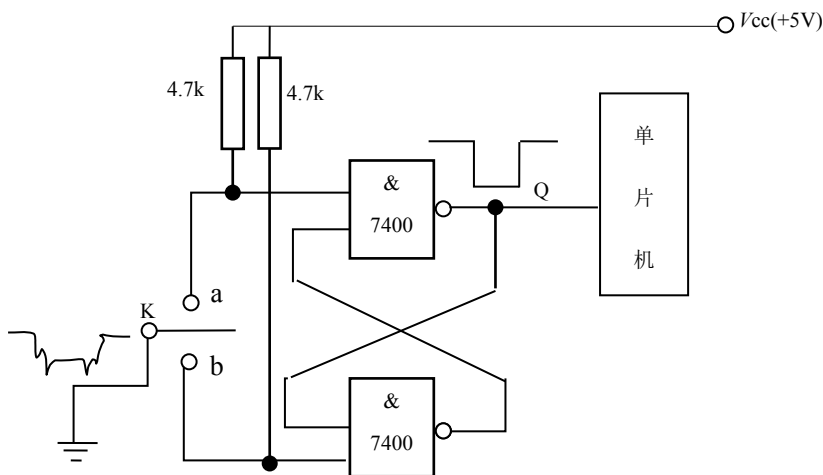


图 6.2 R-S 触发器构成的去抖动电路

电路工作过程如下：按键未按下时， $a=0$ ， $b=1$ ，输出 $Q=1$ ，按键按下时，因按键机械弹性作用的影响，使按键产生抖动，当开关没有稳定到达 b 端时，因与非门 2 输出为 0 反馈到与非门 1 的输入端，封锁了与非门 1，双稳态电路的状态不会改变，输出保持为 1，输出 Q 不会产生抖动的波形。当开关稳定到达 b 端时，因 $a=1$ ， $b=0$ ，使 $Q=0$ ，双稳态电路状态发生翻转。当释放按键式，在开关未稳定到达 a 端时，因 $Q=0$ ，封锁了与非门 2，双稳态电路的状态不变，输出 Q 保持不变消除了后沿抖动波形。当开关稳定到达 b 端时，因

$a=0$, $b=0$, 使得 $Q=1$, 双稳态电路状态发生翻转, 输出 Q 重新返回原状态。可见, 键盘输出经双稳态电路之后, 输出已变为规范的矩形方波。

软件上采取的措施是在检测到有按键按下时, 执行一个 10 ms 左右 (具体时间应视所使用的按键进行调整) 的延时程序后, 再确认该键电平是否仍保持闭合状态电平, 若仍保持闭合状态电平, 则确认该键处于闭合状态。同理, 在检测到该键释放后, 也应采用相同的步骤进行确认, 从而可消除抖动的影响。

(3) 按键编码

一组按键或键盘都要通过 I/O 口查询按键的开关状态。根据键盘结构的不同, 采用不同的编码方法。无论有无编码, 以及采用什么编码, 最后都要转换成为与累加器中数值相对应的键值, 以实现按键功能程序的跳转。

(4) 编制键盘扫描程序

一个完善的键盘控制程序应具备以下功能:

- ① 检测有无按键按下, 并采取硬件或软件措施, 消除键盘按键机械触点抖动的影响。
- ② 有可靠的逻辑处理办法。每次只处理一个按键, 其间对任何按键的操作对系统不产生影响, 且无论一次按键时间有多长, 系统仅执行一次按键功能程序。
- ③ 准确输出按键值 (或键号), 以满足程序跳转指令要求。

6.1.1.2 键盘与单片机的接口

1. 非编码独立式键盘接口

独立式按键是直接由 I/O 口线构成的单个按键电路, 其特点是每个按键单独占用一根 I/O 口线, 每个按键的工作不会影响其他 I/O 口线的状态。独立式按键的典型应用如图 6.3 所示。独立式按键电路配置灵活, 软件结构简单, 但每个按键必须占用一根 I/O 口线, 因此, 在按键较多时, I/O 口线浪费较大, 不宜采用。

图 6.3 所示电路中按键的输入均采用低电平有效, 此外, 上拉电阻器保证了按键断开时, I/O 口线有确定的高电平。当 I/O 内部有上拉电阻器时, 外电路可不接上拉电阻器。

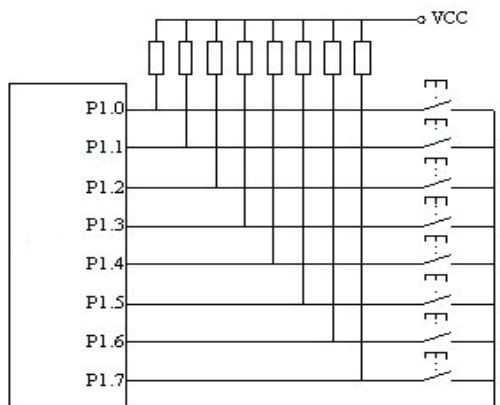


图 6.3 独立式按键电路

独立式按键的软件常采用查询式结构。先逐位查询每根 I/O 口线的输入状态, 如某一

根 I/O 口线输入为低电平，则可确认该 I/O 口线所对应的按键已按下，然后，再转向该键的功能处理程序。图 6.3 中的 I/O 口采用 P1 口，由于独立式键盘比较简单，请读者自行编制相应的软件。

2. 矩阵式键盘接口

单片机系统中，若使用按键较多时，通常采用矩阵式（也称行列式）键盘。

（1）矩阵式键盘的结构及原理

矩阵式键盘由行线和列线组成，按键位于行、列线的交叉点上，其结构如图 6.4 所示。由图 6.4 可知，一个 4×4 的行、列结构可以构成一个含有 16 个按键的键盘，显然，在按键数量较多时，矩阵式键盘较之独立式按键键盘要节省很多 I/O 口。

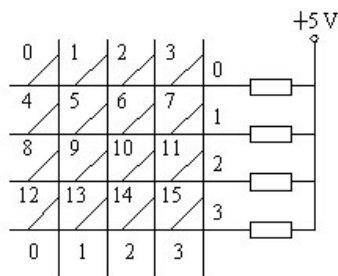


图 6.4 矩阵式键盘结构

矩阵式键盘中，行、列线分别连接到按键开关的两端，行线通过上拉电阻器接到 +5V 上。当无键按下时，行线处于高电平状态；当有键按下时，行、列线将导通，此时，行线电平将由与此行线相连的列线电平决定。这是识别按键是否按下的关键。然而，矩阵键盘中的行线、列线和多个键相连，各按键按下与否均影响该键所在行线和列线的电平，各按键间将相互影响，因此，必须将行线、列线信号配合起来作适当处理，才能确定闭合键的位置。

（2）矩阵式键盘按键的识别

识别按键的方法很多，其中，常见的方法有扫描法和线反转法。下面以图 6.4 中 8 号键的识别为例来说明扫描法识别按键的过程。

按键按下时，与此键相连的行线与列线导通，行线在无键按下时处在高电平。显然，如果让所有的列线也处在高电平，那么，按键按下与否不会引起行线电平的变化，因此，必须使所有列线处在低电平。只有这样，当有键按下时，该键所在的行电平才会由高电平变为低电平。CPU 根据行电平的变化，便能判定相应的行有键按下。8 号键按下时，第 2 行一定为低电平。然而，第 2 行为低电平时，能否肯定是 8 号键按下呢？回答是否定的，因为 9、10、11 号键按下时，同样会使第 2 行为低电平。为进一步确定具体键，不能使所有列线在同一时刻都处在低电平，可在某一时刻只让一条列线处于低电平，其余列线均处于高电平，另一时刻，让下一列处在低电平，依此循环，这种依次轮流每次选通一列的工作方式称为键盘扫描。采用键盘扫描后，再来观察 8 号键按下时的工作过程，当第 0 列处于低电平时，第 2 行处于低电平，而第 1、2、3 列处于低电平时，第 2 行却处在高电平，由此可判定按下的键应是第 2 行与第 0 列的交叉点，即 8 号键。

(3) 键盘的编码

对于独立式按键键盘，因按键数量少，可根据实际需要灵活编码。对于矩阵式键盘，按键的位置由行号和列号唯一确定，因此可分别对行号和列号进行二进制编码，然后将两值合成一个字节，高 4 位是行号，低 4 位是列号。如图 6.4 中的 8 号键，它位于第 2 行，第 0 列，因此，其键盘编码应为 20H。采用上述编码对于不同行的键离散性较大，不利于散转指令对按键进行处理。因此，可采用依次排列键号的方式对按键进行编码。以图 6.4 中的 4×4 键盘为例，可将键号编码为：01H、02H、03H、……、0EH、0FH、10H 等 16 个键号。编码相互转换可通过计算或查表的方法实现。

6.1.1.3 键盘的工作方式

对键盘的响应取决于键盘的工作方式，键盘的工作方式应根据实际应用系统中单片机的工作状况而定，其选取的原则是既要保证 CPU 能及时响应按键操作，又不要过多占用单片机的工作时间。通常，键盘的工作方式有三种，即编程扫描、定时扫描和中断扫描。

1. 编程扫描方式

编程扫描方式是利用单片机完成其他工作的空余时间，调用键盘扫描子程序来响应键盘输入的要求。在执行键功能程序时，单片机不再响应键输入要求，直到单片机重新扫描键盘为止。

键盘扫描程序一般应包括以下内容：

- (1) 判别有无键按下。
- (2) 键盘扫描取得闭合键的行、列值。
- (3) 用算法或查表法得到键值。
- (4) 判断闭合键是否释放，如没释放则继续等待。
- (5) 将闭合键键号保存，同时转去执行该闭合键的功能。

2. 定时扫描方式

定时扫描方式就是每隔一段时间对键盘扫描一次，它利用单片机内部的定时器产生一定时间（例如 10ms）的定时，当定时时间到就产生定时器溢出中断，CPU 响应中断后对键盘进行扫描，并在有键按下时识别出该键，再执行该键的功能程序。

3. 中断扫描方式

采用上述两种键盘扫描方式时，无论是否按键，单片机都要定时扫描键盘，而单片机应用系统工作时，并非经常需要键盘输入，因此，单片机经常处于空扫描状态，为提高单片机工作效率，可采用中断扫描工作方式。其工作过程如下：

当无键按下时，单片机处理自己的工作，当有键按下时产生中断请求，单片机转去执行键盘扫描子程序，并识别键号。

图 6.5 所示是一种简易键盘接口电路，该键盘是由单片机的 P1 口的高、低字节构成的 4×4 键盘。键盘的列线与 P1 口的高 4 位相连，键盘的行线与 P1 口的低 4 位相连，因此，P1.4~P1.7 是键输出线，P1.0~P1.3 是扫描输入线。图 6.5 所示电路中的 4 输入与非门用于产生按键中断，其输入端与各列线相连，在通过上拉电阻器接到+5V 电源，输出端接至 CPU

的外部中断输入端 $\overline{\text{INT0}}$ 。具体工作如下：当键盘无键按下时，与门各输入端均为高电平，保持输出端为高电平；

当有键按下时， $\overline{\text{INT0}}$ 端为低电平，向 CPU 申请中断，若 CPU 开放外部中断，则会响应中断请求，转去执行键盘扫描子程序。

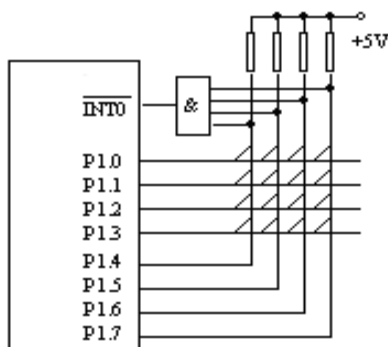


图 6.5 中断方式键盘接口电路

6.1.2 LED 显示接口设计

LED (Light Emission Diode) 是发光二极管的缩写。LED 数码管成本低廉、配置灵活，与单片机接口简单，在单片机系统中获得广泛地应用。

6.1.2.1 LED 数码管概述

1. LED 数码管结构

LED 数码管是由发光二极管组合显示字符的显示器件。常用的 LED 数码管有 7 段、8 段等类型。七段 LED 数码管的内部电路结构及引封装形式如图 6.6 所示。1 个数码管由 8 个发光二极管组成，其中 7 个发光二极管构成 a~g 共 7 段用于显示字符，另 1 个用于显示小数点。故通常称之为 7 段（也有称作 8 段）发光二极管数码显示器。

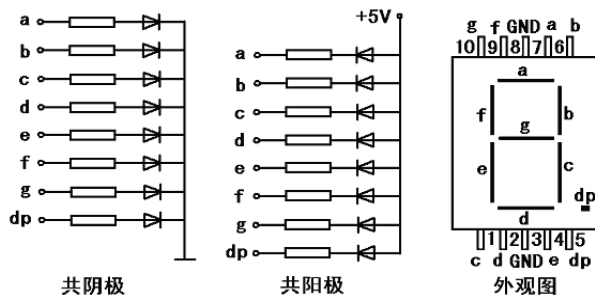


图 6.6 7 段 LED 数码管内部电路结构及引脚封装形式

从内部电路上看，数码管又可分为共阴极接法（8 个发光二极管的阴极接在一起）和共阳极接法（8 个发光二极管的阳极接在一起），如图 6.6 所示。通过对公共端（COM）接

地或接高电平的控制,可使共阴极或共阳极数码管根据由 a~g 引脚输入的代码来显示数字或符号。

2. LED 数码管的显示原理

为了让数码管显示数字或符号,要为 LED 显示器提供显示段码(或称字形代码),这些代码是使显示器显示字形的,所以也称之为字形代码、段选码。7 段数码管用组成一个“8”字形字符的 7 段,再加上 1 个小数点位,共计 8 段,因此提供给 LED 显示器的显示段码为 1 个字节。各段码位的对应关系如表 6.1 所列。

表 6.1 段码与字节中各位对应关系

段码位	D7	D6	D5	D4	D3	D2	D1	D0
显示段	dp	g	f	e	d	c	b	a

根据 LED 数码管结构可知,如果希望显示“8”字,那么除了“dp”管不要点亮以外,其余管全部点亮。同理,如果要显示“1”,那么只需要 b、c 两个发光二极管点亮。对于共阳极结构,就是要把公共端 COM 接到电源正极,而 b、c 两个负极分别经过一个限流电阻器后接低电平;对于共阴极结构,就是要把公共端 COM 接低电平(电源负极),而 b、c 两个正极分别经一个限流电阻器后接到高电平。单片机系统扩展 LED 数码管时多用共阳极 LED。共阳极数码管每个段笔画是用低电平(“0”)点亮的,要求驱动功率很小;而共阴极数码管段笔画是用高电平(“0”)点亮的,要求驱动功率较大。通常每个段笔画要串一个数百欧姆的降压电阻器。

应用中要将一个 8 位并行段选码送至 LED 显示器对应的引脚,送入的段选码不同,显示的数字或字符也不同。共阴极与共阳极的段选码互为反码,如表 6.2 所示。

表 6.2 LED 数码管段选表

字 形	共阴极段码	共阳极段码	字 形	共阴极段码	共阳极段码
0	3FH	C0H	C	39H	C6H
1	06H	F9H	d	5EH	A1H
2	5BH	A4H	E	79H	86H
3	4FH	B0H	F	71H	8EH
4	66H	99H	P	73H	8CH
5	6DH	92H	U	3EH	C1H
6	7DH	82H	Γ	31H	CEH
7	07H	F8H	Y	6EH	91H
8	7FH	80H	H	76H	89H
9	6FH	90H	L	38H	C7H
A	77H	88H	灭	00H	FFH
b	7CH	83H			

注意

这种规定和定义并非一成不变的,在实际应用中,为了减少走线交叉,便于电路板布线,设计者可自行定义 8 段 LED 数码管的引脚符号,并根据其工作原理编制相应的“段码表”以及设计与之相匹配的连接电路。

6.1.2.2 LED 数码管的工作方式

1. 静态显示方式

所谓静态显示，就是当显示某一个数字时，代表相应笔画的发光二极管恒定发光，例如8段数码管的a、b、c、d、e、f笔段亮时显示数字“0”；b、c亮时显示“1”；a、b、d、e、g亮时显示“2”等。其显示方法比较简单，只要将显示段码送至段码口，并把位控字送至位控口即可。所用指令为：

```
MOV DPTR, #SEGPORT; 指向段码口
MOV A, #SEG;          取显示段码
MOVX @DPTR, A;        输出段码
MOV DPTR, #BITPORT; 指向位控口
MOV A, #BIT;          取位控字
MOVX @DPTR, A;        输出位控字
```

静态显示方式的数码管中各位相互独立，每位LED数码管需要一个8位的I/O口输出相应的字形码，每位的显示字符一经确定，相应锁存的输出将维持不变。因此，静态显示的优点是显示稳定，在驱动电流一定的情况下显示的亮度高，编程容易，管理也较为简单。但是，由于静态显示方式占用I/O口资源较多，在显示位数较多时，会大大增加硬件电路成本（使用元器件较多，每一位都需要一个驱动器，每一段都需要一个限流电阻器，连接线多）。只适合于显示位数较少的场合。

2. 动态显示

动态显示是一位一位地轮流点亮各位数码管，这种逐位点亮显示器的方式称为位扫描。通常，各位数码管的段选线相应并联在一起，由一个8位的I/O口控制；各位的位选线（公共阴极或阳极）由另外的I/O口线控制。动态方式显示时，各数码管分时轮流选通，要使其稳定显示，必须采用扫描方式，即在某一时刻只选通一位数码管，并送出相应的段码，在另一时刻选通另一位数码管，并送出相应的段码。依此规律循环，即可使各位数码管显示将要显示的字符。虽然这些字符是在不同的时刻分别显示，但由于人眼存在视觉暂留效应，只要每位显示间隔足够短就可以给人以同时显示的感觉。

采用动态显示方式比较节省I/O口，硬件电路也较静态显示方式简单，但其亮度不如静态显示方式，而且在显示位数较多时，CPU要依次扫描，占用CPU较多的时间。

用8051系列单片机构建数码管动态显示系统时，常采用8255可编程I/O扩展接口，其显示电路如图6.7所示。

图中，数码管采用共阴极LED，8255的A口线经过8路驱动电路后接至数码管的各段。当A口线输出“1”时，驱动数码管发光。8255的C口线经过6路驱动电路后接至数码管的公共端。当C口线输出“0”时，选通相应位的数码管发光。

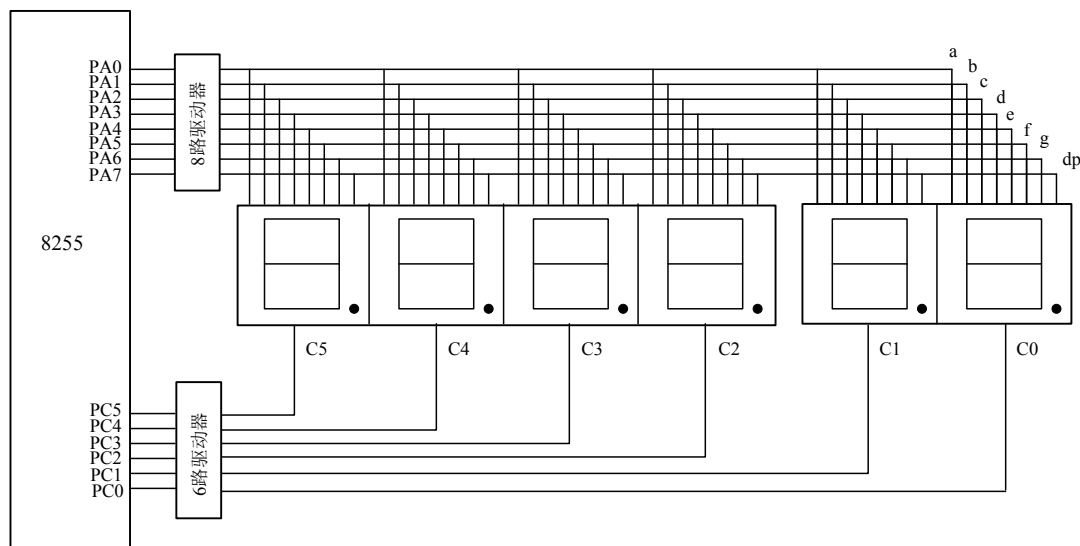


图 6.7 8255 构成的六位数码管动态显示电路

A 口、C 口应定义为基本输出，分别控制数码管的段码（段驱动端）和位码（公共端），B 口未用，可定义为基本输入。此时，命令寄存器中的 PA=1，PB=0，PC1=1，PC2=1。因不用 A、B 口中断，也不用定时/计数器，故 IEA=0，IEB=0，TM1=1，TM2=0。由此可得命令字为：01001101B=4DH。编程时的步骤如下：

```
MOV DPTR, #CWR ; 选中 8155 的命令寄存器
```

```
MOV A, #4DH ; 命令字送 A
```

```
MOVX @DPTR, A ; 命令字写入命令寄存器
```

比较静态显示与动态显示可知，二者功能完全一样。前者数码管较亮，且显示程序占用 CPU 的时间较少，但其硬件电路复杂，占用单片机口线多，成本高；后者硬件电路相对简单，成本较低，但其数码管显示亮度偏低，且采用动态扫描方式，显示程序占用 CPU 的时间较多。具体应用时，应根据实际情况，选用合适的显示方式。

同样都是动态扫描显示，采用不断调用子程序的方式实现动态扫描显示，亮度相对较高，CPU 效率较低；采用定时器中断（20 ms 中断一次）的方式实现动态扫描显示，亮度较低，CPU 效率相对较高。

针对数码管显示亮度偏低的情况，可采用提高扫描速度（如由 20 ms 改为 10 ms），或适当延长单只数码管导通的时间（如导通延时时间由 1 ms 改为 2 ms）等措施来弥补，但其带来的后果是显示程序占用 CPU 的时间更多，导致 CPU 利用率更低。

6.1.3 LCD 显示接口设计

液晶显示器简称 LCD（Liquid Crystal Diodes）是利用液晶经过处理后能够改变光线传输方向的特性，达到显示字符或者图形的目的。其特点是体积小、重量轻、功耗极低、显示内容丰富等特点，在单片机应用系统中有着日益广泛的应用。

6.1.3.1 液晶显示简介

液晶显示的原理是利用液晶的物理特性，通过电压对其显示区域进行控制，有电就有显示，这样即可以显示出图形。液晶显示器具有厚度薄、适用于大规模集成电路直接驱动、易于实现全彩色显示的特点，目前已经被广泛应用在便携式电脑、数字摄像机、PDA 移动通信工具等众多领域。

液晶显示器的分类方法有很多种，通常可按其显示方式分为段式、字符式、点阵式等。除了黑白显示外，液晶显示器还有多灰度有彩色显示等。如果根据驱动方式来分，可以分为静态驱动（Static）、单纯矩阵驱动（Simple Matrix）和主动矩阵驱动（Active Matrix）三种。按排列形状分：字段形、点阵字符形和点阵图形。字段形的 LCD 广泛用于电子表、数字仪表、计算器中；点阵字符形的 LCD 显示字母、数字、符号，它是由 5×7 或 5×10 点阵组成，广泛应用在单片机应用系统中；点阵图形 LCD 用于笔记本电脑和彩色电视机等设备中。

6.1.3.2 点阵字符形 LCD1602 简介

现在已将 LCD 控制器、驱动器、RAM、ROM 和 LCD 显示器用 PCB 连接到一起，称为液晶显示模块 LCM（LCd Module）。字符形液晶显示模块是一种专门用于显示字母、数字、符号等点阵式 LCD，目前常用 16×1，16×2，20×2 和 40×2 行等的模块。现简要介绍 LCD1602 的用法。

1. LCD1602 的引脚功能

1602LCD 分为带背光和不带背光两种，其控制器大部分为 HD44780，带背光的比不带背光的厚，是否带背光在应用中并无差别，两者尺寸差别如图 6.8 所示。

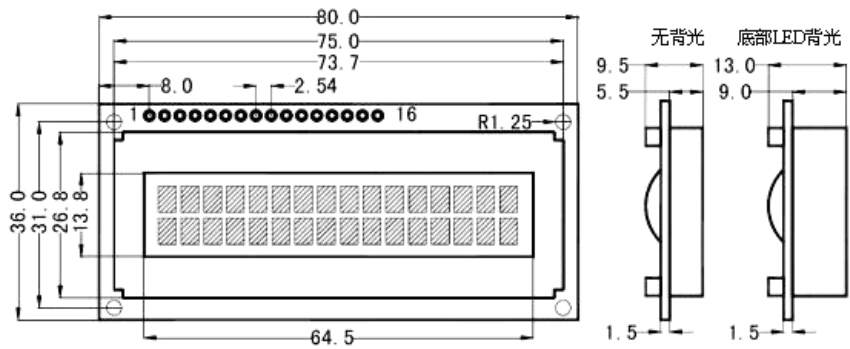


图 6.8 1602LCD 尺寸图

1602LCD 采用标准的 14 脚（无背光）或 16 脚（带背光）接口，各引脚接口说明如表 6.3 所示。

表 6.3 引脚接口说明表

编 号	符 号	引脚说明	编 号	符 号	引脚说明
1	VSS	电源地	9	D2	数据
2	VDD	电源正极	10	D3	数据
3	VL	液晶显示偏压	11	D4	数据
4	RS	数据/命令选择	12	D5	数据
5	R/W	读/写选择	13	D6	数据
6	E	使能信号	14	D7	数据
7	D0	数据	15	BLA	背光源正极
8	D1	数据	16	BLK	背光源负极

第 1 脚：VSS 为地电源。

第 2 脚：VDD 接 5V 正极电源。

第 3 脚：VL 为液晶显示器对比度调整端，接正极电源时对比度最弱，接地时对比度最高，对比度过高时会产生“鬼影”，使用时可以通过一个 10k Ω 的电位器调整对比度。

第 4 脚：RS 为寄存器选择，高电平时选择数据寄存器、低电平时选择指令寄存器。

第 5 脚：R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 R/W 共同为低电平时可以写入指令或者显示地址，当 RS 为低电平 R/W 为高电平时可以读忙信号，当 RS 为高电平 R/W 为低电平时可以写入数据。

第 6 脚：E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。

第 7 脚~第 14 脚：D0~D7 为 8 位双向数据线。

第 15 脚：背光源正极。

第 16 脚：背光源负极。

2. LCD1602 的指令说明

(1) 寄存器的选择

表 6.4 寄存器的选择

RS	R/W*	操 作
0	0	命令寄存器写入
0	1	忙标志和地址计数器读出
1	0	数据寄存器写入
1	1	数据寄存器读出

(2) 控制命令

1602 液晶模块内部的控制器共有 11 条控制命令，如表 6.5 所列。

表 6.5 控制命令表

序 号	命 令	引 脚		状 态 字							
		RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清除显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	设置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*

(续表)

序 号	命 令	引 脚		状 态 字							
		RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
6	设置功能	0	0	0	0	1	DL	N	F	*	*
7	设置字符发生存储器地址	0	0	0	1	字符发生存储器地址					
8	设置数据存储器地址地址	0	0	1	显示数据存储器地址						
9	读忙标志或地址	0	1	BF	计数器地址						
10	写数据到 CGRAM 或 DDRAM	1	0	要写的数字内容							
11	从 CGRAM 或 DDRAM 读数据	1	1	读出的数字内容							

1602 液晶模块的读写操作、屏幕和光标的操作都是通过命令编程来实现的。

上表命令说明：

- ① 清除显示：指令码 01H，光标复位到地址 00H 位置。
- ② 光标返回：光标返回到地址 00H，显示回到原始位置。
- ③ 设置输入模式：I/D：光标移动方向，高电平右移，低电平左移；S：屏幕上所有文字是否左移或者右移。高电平表示有效，低电平则无效。

④ 显示开/关控制。

D：控制整体显示的开与关，D=1 开显示，D=0 关显示

C：控制光标的开与关，C=1 有光标，C=0 无光标

B：控制光标是否闪烁，B=1 闪烁，B=0 不闪烁。

⑤ 光标或显示移位

S/C：滚动对象的选择，高电平时移动显示的文字，低电平时移动光标。

S/C=1 画面滚动 S/C=0 光标滚动

R/L：滚动方向的选择

R/L=1 向右滚动 R/L=0 向左滚动

⑥ 设置功能

DL：DL=0 为 4 位总线，DL=1 为 8 位总线

N：N=0 为单行显示，N=1 双行显示

F：F=0 显示 5×7 的点阵字符，F=1 时显示 5×10 的点阵字符。

⑦ 字符发生器 RAM 地址设置。

⑧ DDRAM 地址设置。

⑨ 读忙信号和光标地址

BF：忙标志位，高电平表示忙，此时模块不接收命令或者数据，如为低，表示不忙。

⑩ 写数据。

⑪ 读数据。

(3) 1602LCD 的 RAM 地址映射及标准字库表

液晶显示模块是一个慢显示器件，所以在执行每条指令之前一定要确认模块的忙标志为低电平，表示不忙，否则此指令失效。要显示字符时，要先输入显示字符地址，也就是告诉模块在哪里显示字符，图 6.9 所示是 1602 的内部显示地址。

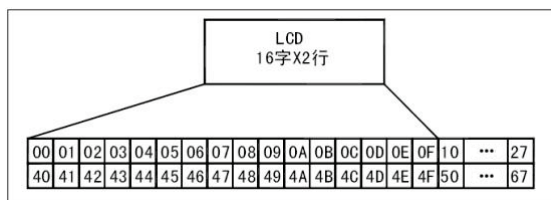


图 6.9 1602LCD 内部显示地址

例如第二行第一个字符的地址是 40H, 那么是否直接写入 40H 就可以将光标定位在第二行第一个字符的位置呢? 这样不行, 因为写入显示地址时, 要求最高位 D7 恒定为高电平 1, 所以实际写入的数据应该是 $01000000B(40H) + 10000000B(80H) = 11000000B(C0H)$ 。

在对液晶模块的初始化中, 要先设置其显示模式, 在液晶模块显示字符时, 光标是自动右移的, 无需人工干预。每次输入指令前都要判断液晶模块是否处于忙的状态。

1602 液晶模块内部的字符发生存储器 (CGROM) 已经存储了 160 个不同的点阵字符图像, 如图 6.10 所示, 这些字符有阿拉伯数字、英文字母的大小写、常用的符号和日文假名等, 每一个字符都有一个固定的代码, 比如大写的英文字母 “A” 的代码是 01000001B(41H), 显示时模块把地址 41H 中的点阵字符图形显示出来, 我们就能看到字母 “A”。

Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits	Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	(1)				0	1	2	3	4	5	6	7	8	9	A	B	C
xxxx0001	(2)				!	@	#	\$	%	&	'	()	*	+	=	-
xxxx0010	(3)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx0011	(4)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx0100	(5)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx0101	(6)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx0110	(7)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx0111	(8)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1000	(1)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1001	(2)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1010	(3)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1011	(4)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1100	(5)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1101	(6)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1110	(7)				"	#	\$	%	&	'	()	*	+	=	-	.
xxxx1111	(8)				"	#	\$	%	&	'	()	*	+	=	-	.

Note: The user can specify any pattern for character-generator RAM.

图 6.10 字符代码与图形对应图

(4) 1602LCD 的一般初始化（复位）过程



图 6.11 1602LCD 的一般初始化（复位）过程

编程时可按如下流程写程序：

延时 15ms

写指令 38H（不检测忙信号）

延时 5ms

写指令 38H（不检测忙信号）

延时 5ms

写指令 38H（不检测忙信号）

以后每次写指令、读/写数据操作均需要检测忙信号

写指令 38H：显示模式设置

写指令 08H：显示关闭

写指令 01H：显示清屏

写指令 06H：显示光标移动设置

写指令 0CH：显示开及光标设置

6.2 学习实例

实例一 用 LED 数码管循环显示 0~9

1. 实例说明

将 0~9 这 10 个数字的段码存入数组，每隔一段时间取出一个数字的段码从 P0 口输出。

2. 仿真电路

用 LED 数码管循环显示 0~9 仿真电路如图 6.12 所示。

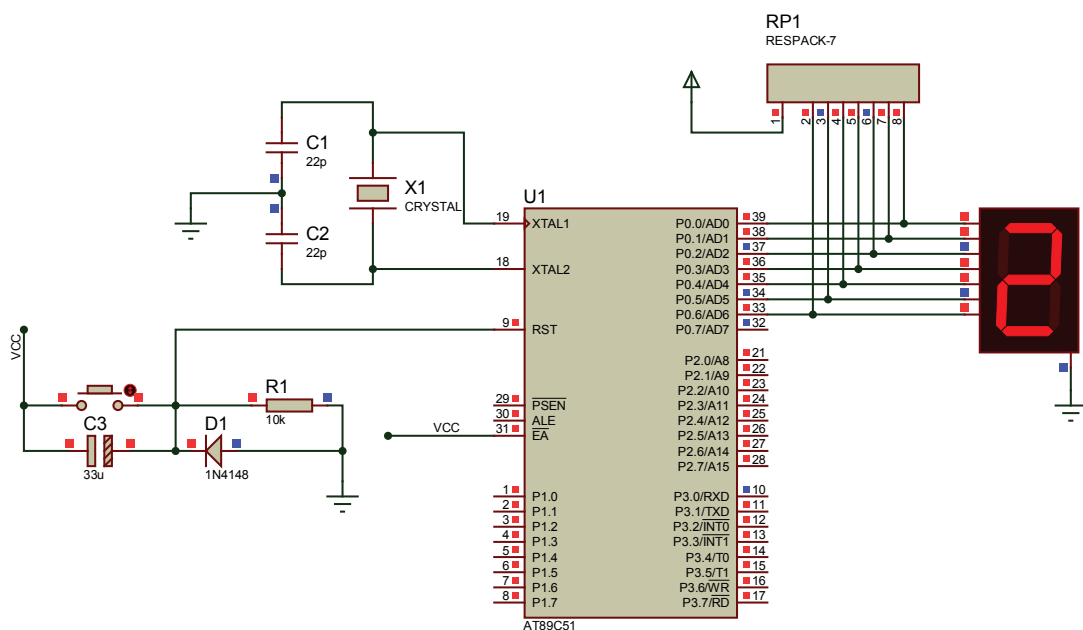


图 6.12 用 LED 数码管循环显示 0~9 仿真电路

3. 程序设计

```
#include<reg51.h> //包含单片机寄存器的头文件
unsigned char code segcode[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,
0x6F};
void delay(void)
{
    unsigned int i,j;
    for(i=0;i<400;i++)
        for(j=0;j<255;j++);
}
void main(void)
{
    unsigned char k;
    while(1)
    {
        for(k=0;k<10;k++)
        {
            P0=segcode[k]; //P0 口送数字段码
            delay();
        }
    }
}
```

实例二 用LED数码管动态显示“HELLO”

1. 实例说明

要求用8255扩展一并行接口，其中8255的A口输出段码，B口输出位选码，由7段显示器显示“HELLO”字样。

2. 仿真电路

用LED数码管动态显示“HELLO”仿真电路如图6.13所示。

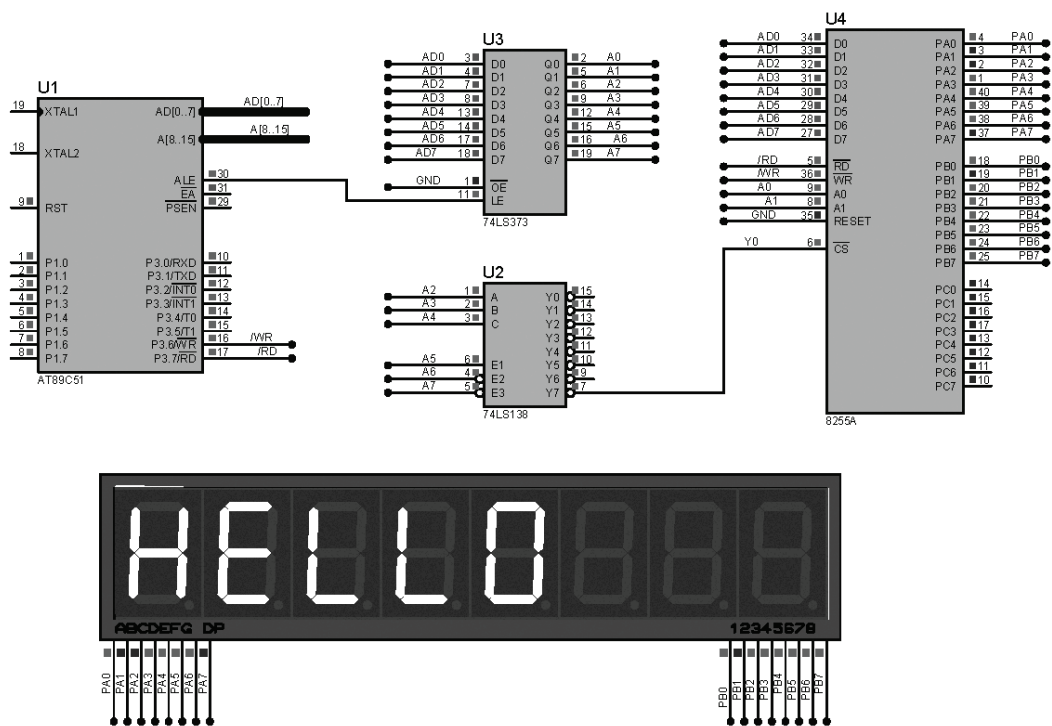


图 6.13 用LED数码管动态显示“HELLO”仿真电路

3. 程序设计

```
#include<reg51.h>
#include<absacc.h>
#define PA XBYTE[0xff3c]
#define PB XBYTE[0xff3d]
#define PC XBYTE[0xff3e]
#define PK XBYTE[0xff3f]
unsigned char dispcode[]={0x76,0x79,0x38,0x38,0x3f};
```

```
unsigned char k[]={0xfe,0xfd,0xfb,0xf7,0xef};
void delay100(int n)
{
    while(n--)
    {
        int i,j;
        for(i=0;i<10;i++)
            for(j=0;j<10;j++)
                ;
    }
}
void main()
{
    PK=0x80; //8255 初始化, A、B 口均为方式 0 输出
    while(1)
    {
        int i;
        for(i=0;i<5;i++)
        {
            PA=dispcode[i];
            PB=k[i];
            delay100(10);
            PB=0xff; //清屏
        }
    }
}
```

实例三 数码时钟设计

1. 实例说明

要求设计一个数码时钟, 由 74LS138 选择数码管的每一位, 74HC573 锁存段码。

2. 仿真电路

数码时钟设计仿真电路如图 6.14 所示。

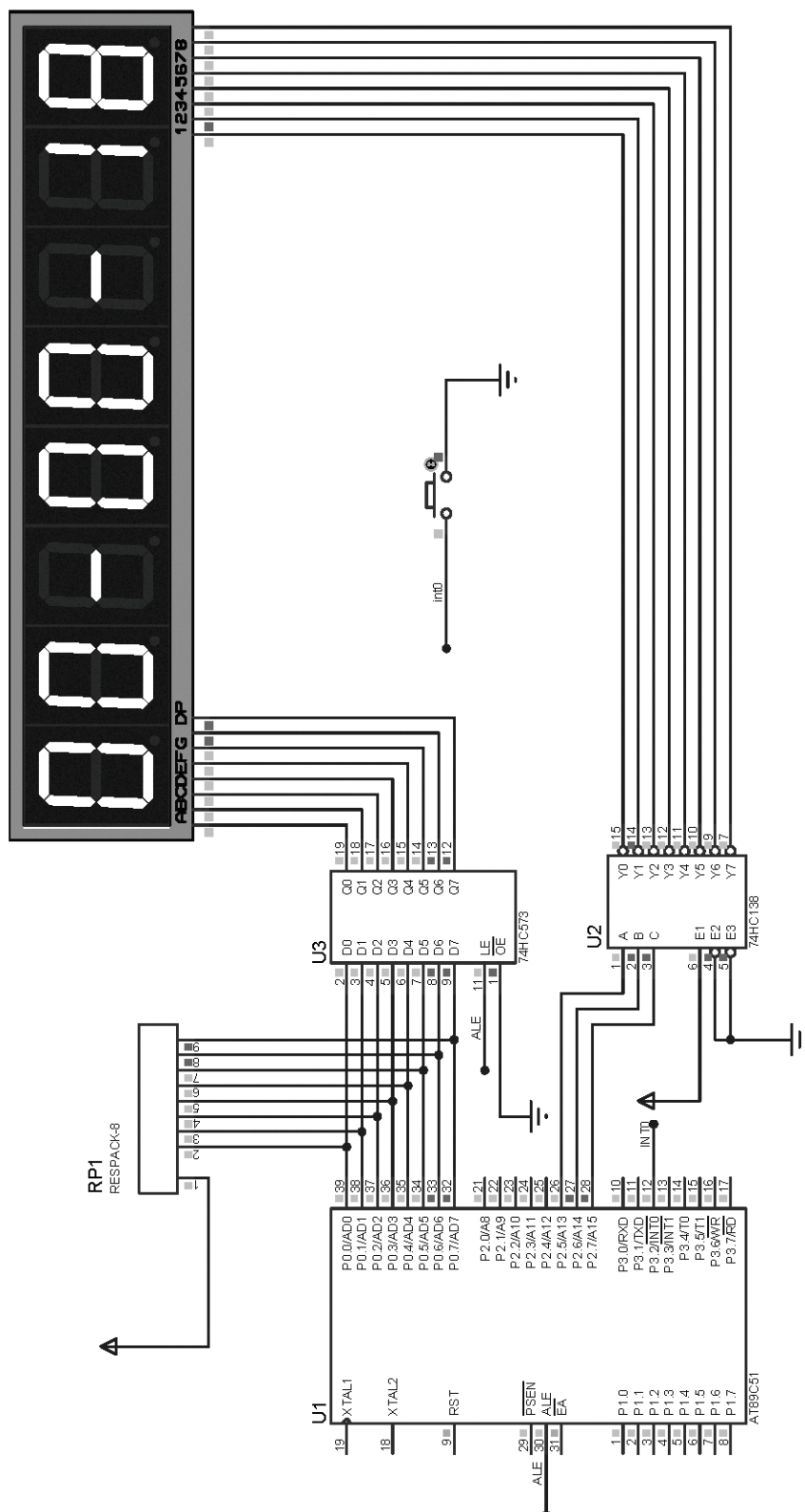


图 6.14 数码时钟设计仿真电路

3. 程序设计

```
#include < reg51.h >
sbit LS138A=P2^5; //定义 138 管脚
sbit LS138B=P2^6;
sbit LS138C=P2^7;
unsigned char h,m,s,c,LedOut[8];
unsigned char code Disp_Tab[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0x7f,0x6f};
/*****
purpose: 系统初始化
*****/
void system_Ini()
{
    TMOD = 0x01; //选择模式 1 16 位计数 最大计数 65536
    TH0 = (65536-50000)/256;
    TL0 = (65536-50000)%256;
    IE = 0x8B; //允许中断
    IT0=1;
    TR0=1; //启动 T0
}
void delay(unsigned int i)
{
    char j;
    for(i; i > 0; i--)
        for(j = 200; j > 0; j--);
}
void display()
{
    while(1)
    {
        unsigned char i;
        LedOut[0]=Disp_Tab[h/10];
        LedOut[1]=Disp_Tab[h%10];
        LedOut[2]=0x40;
        LedOut[3]=Disp_Tab[m/10];
        LedOut[4]=Disp_Tab[m%10];
        LedOut[5]=0x40;
        LedOut[6]=Disp_Tab[s/10];
        LedOut[7]=Disp_Tab[s%10];
        for(i=0;i<8;i++)
        {P0=LedOut[i];
        switch(i)
        {
            case 0:LS138A=0; LS138B=0; LS138C=0; break;
            case 1:LS138A=1; LS138B=0; LS138C=0; break;
            case 2:LS138A=0; LS138B=1; LS138C=0; break;
            case 3:LS138A=1; LS138B=1; LS138C=0; break;
            case 4:LS138A=0; LS138B=0; LS138C=1; break;
            case 5:LS138A=1; LS138B=0; LS138C=1; break;
```

```

        case 6:LS138A=0; LS138B=1; LS138C=1; break;
        case 7:LS138A=1; LS138B=1; LS138C=1; break;
    }
    delay(150);
    P0=0x00;
}
}
}
/*****主函数*****/
void main()
{
    system_Ini()    ;
    h=0;
    m=0;
    s=0;
    c=0;
    display();
}
/*****
T0(50ms)中断
*****/
void T0zd(void) interrupt 1    // 定时器0的中断服务
{
    c++;
    TH0=(65536-50000)/256;
    TL0=(65536-50000)%256;
    if(c==20)
    {
        c=0;
        s++;
        if(s==60)
        {
            s=0;
            m++;
        }
        if(m==60)
        {
            m=0;
            h++;
        }
        if(h==23)
        {
            h=0;
        }
    }
}
}
void INT0zd(void) interrupt 0// 外部中断0的中断服务
{
    s=0;
    h=0;

```

```

m=0;
c=0;
}

```

实例四 独立式键盘控制步进电动机正、反转

1. 实例分析

要求设计一个控制步进电动机正、反转的系统，按下 S1 反转，按下 S2 正转，按下 S2 停止。驱动用 ULN2003A，请查阅 ULN2003A、步进电动机资料。

2. 仿真电路

独立式键盘控制步进电动机正、反转仿真电路如图 6.15 所示。

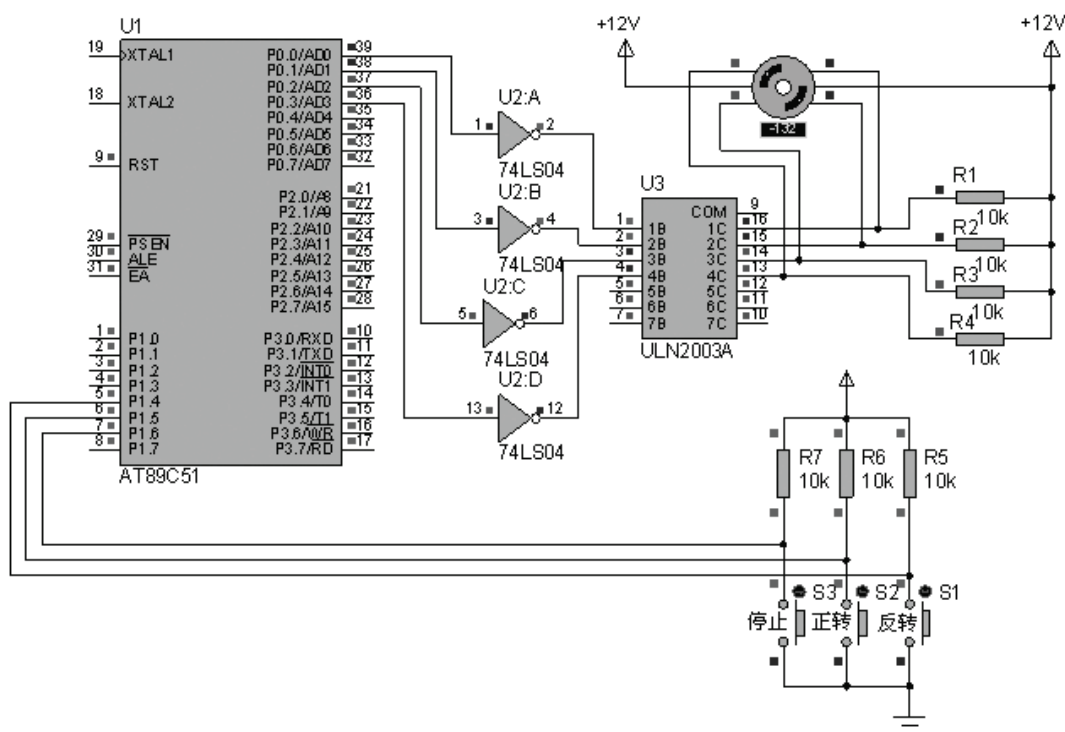


图 6.15 独立式键盘控制步进电动机正、反转仿真电路

3. 程序设计

```

#include<reg51.h>           //包含 51 单片机寄存器定义的头文件
sbit S1=P1^4;               //将 S1 位定义为 P1.4 引脚
sbit S2=P1^5;               //将 S2 位定义为 P1.5 引脚
sbit S3=P1^6;               //将 S3 位定义为 P1.6 引脚
unsigned char keyval;       //储存按键值
unsigned char ID;           //储存功能标号
/*****
函数功能：软件消抖延时（约 50ms）

```

```

*****/
void delay(void)
{
    unsigned char i,j;
    for(i=0;i<150;i++)
        for(j=0;j<100;j++)
            ;
}
/*****
函数功能：步进电动机转动延时，延时越长，转速越慢
*****/
void motor_delay(void)
{
    unsigned int i;
    for(i=0;i<2000;i++)
        ;
}
/*****
函数功能：步进电动机正转
*****/
void forward( )
{
    P0=0xfc;          //P0 口低四位脉冲 1100
    motor_delay();
    P0=0xf6;          //P0 口低四位脉冲 0110
    motor_delay();
    P0=0xf3;          //P0 口低四位脉冲 0011
    motor_delay();
    P0=0xf9;          //P0 口低四位脉冲 1001
    motor_delay();
}
/*****
函数功能：步进电动机反转
*****/
void backward()
{
    P0=0xfc;          //P0 口低四位脉冲 1100
    motor_delay();
    P0=0xf9;          //P0 口低四位脉冲 1001
    motor_delay();
    P0=0xf3;          //P0 口低四位脉冲 0011
    motor_delay();
    P0=0xf6;          //P0 口低四位脉冲 0110
    motor_delay();
}
/*****
函数功能：步进电动机停转
*****/
void stop(void)
{
    P0=0xff ;          //停止输出脉冲

```

```

}
/*****
函数功能：主函数
*****/
void main(void)
{
    TMOD=0x01;           //使用定时器 T0 的模式 1
    EA=1;                //开总中断
    ET0=1;               //定时器 T0 中断允许
    TR0=1;               //启动定时器 T0
    TH0=(65536-500)/256;  //定时器 T0 赋初值，每计数 200 次（217 微秒）发送一次中断
    TL0=(65536-500)%256;  //定时器 T0 赋初值
    keyval=0;             //按键值初始化为 0，什么也不做
    ID=0;
    while(1)
    {
        switch(keyval)    //根据按键值 keyval 选择待执行的功能
        {
            case 1:forward(); //按键 S1 按下，正转
                break;
            case 2:backward(); //按键 S2 按下，反转
                break;
            case 3:stop();     //按键 S3 按下，停转
                break;
        }
    }
}
/*****
函数功能：定时器 T0 的中断服务子程序
*****/
void Time0_serve(void) interrupt 1 using 1
{
    TR0=0;                //关闭定时器 T0
    if((P1&0xf0)!=0xf0)    //第一次检测到有键按下
    {
        delay();          //延时一段时间再去检测
        if((P1&0xf0)!=0xf0) //确实有键按下
        {
            if(S1==0)      //按键 S1 被按下
                keyval=1;
            if(S2==0)      //按键 S2 被按下
                keyval=2;
            if(S3==0)      //按键 S3 被按下
                keyval=3;
        }
    }
    TH0=(65536-200)/256;   //定时器 T0 的高 8 位赋初值
    TL0=(65536-200)%256;   //定时器 T0 的低 8 位赋初值
    TR0=1;                 //启动定时器 T0
}

```

实例五 矩阵式键盘按键值的数码管显示

1. 实例说明

要求用单片机的 P2 口作为键盘接口接 4×4 键盘，由 P0 口作为显示接口，七段显示器显示 16 个按键对应的键值 0~9，A~F。

2. 仿真电路

矩阵式键盘按键值的数码管显示仿真电路如图 6.16 所示。

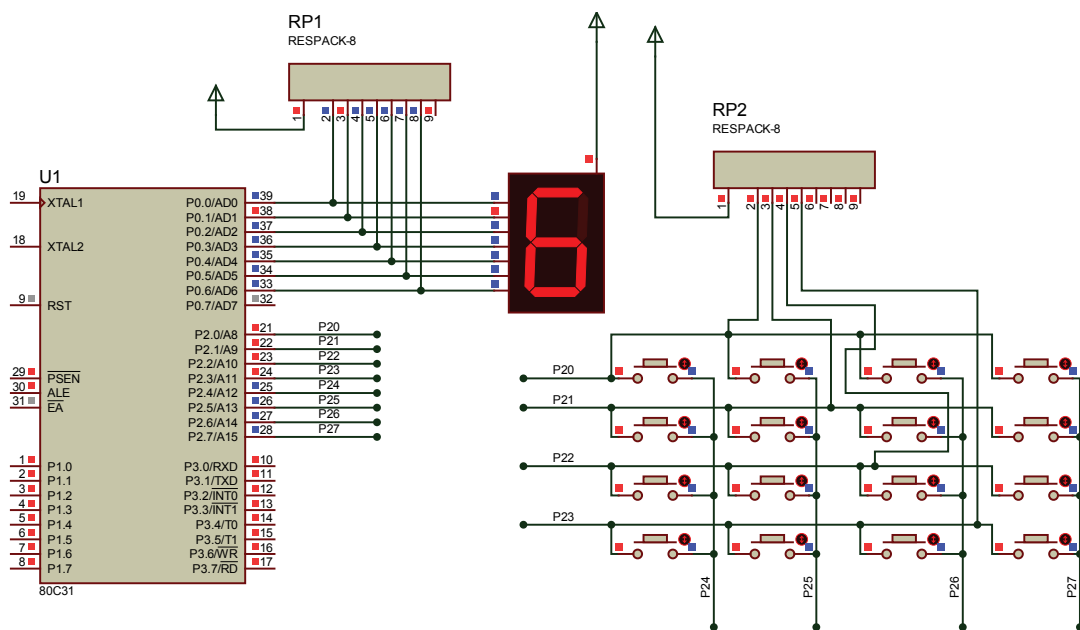


图 6.16 矩阵式键盘按键值的数码管显示仿真电路

3. 程序设计

```
#include <REG51.h> // MCS-51 头文件声明
#define uchar unsigned char // 定义无符号变量简写式
#define uint unsigned int //
char
led[]={0xC0,0xF9,0xa4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
sfr KEY=0xa0; // 定义 P2 口为键盘接口
sfr Disp=0x80; // P0 口为显示接口

void Delay(uchar); // 延时程序声明（单位为毫秒）
uchar key();
uchar Key() // 键盘处理函数
{
    uchar a,b,c; // 定义 3 个变量
```

```
KEY=0x0f; //键盘口置 00001111
if (KEY!=0x0f) //查寻键盘口的值是否变化
{
    Delay(20); //延时 20 毫秒
    if (KEY!=0x0f) //有键按下处理
    {
        a=KEY; //键值放入寄存器 a
    }
    KEY=0xf0; //将键盘口置为 11110000
    c=KEY; //将第二次取得值放入寄存器 c
    a=a|c; //将两个数据熔合
    switch(a) //对比数据值
    {
        case 0xee: b = 0x00; break; //对比得到的键值给 b 一个应用数据
        case 0xde: b = 0x01; break;
        case 0xbe: b = 0x02; break;
        case 0x7e: b = 0x03; break;
        case 0xed: b = 0x04; break;
        case 0xdd: b = 0x05; break;
        case 0xbd: b = 0x06; break;
        case 0x7d: b = 0x07; break;
        case 0xeb: b = 0x08; break;
        case 0xdb: b = 0x09; break;
        case 0xbb: b = 0x0a; break;
        case 0x7b: b = 0x0b; break;
        case 0xe7: b = 0x0c; break;
        case 0xd7: b = 0x0d; break;
        case 0xb7: b = 0x0e; break;
        case 0x77: b = 0x0f; break;
        default: break; //键值错误处理
    }
}
return(b); //将 b 作为返回值
}
void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //1 毫秒延时
    }
}
void main()
{
    while(1)
    {
        Disp=led[key()];
    }
}
```


实例六 矩阵式键盘按键值的 LCD 显示

1. 实例说明

要求用单片机的 P2 口作为键盘接口接 4×4 键盘，由 P0 口作为显示接口，LCD 显示器显示 16 个按键对应的键值 0~9，A~F。

2. 仿真电路

矩阵式键盘按键值的 LCD 显示仿真电路如图 6.17 所示。

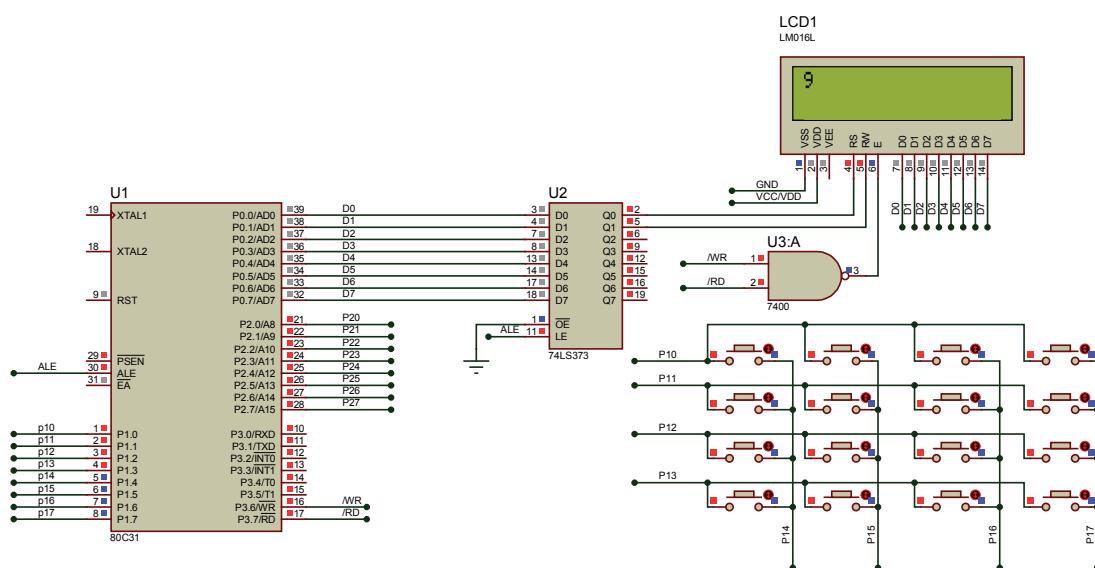


图 6.17 矩阵式键盘按键值的 LCD 显示仿真电路

3. 程序设计

```
#include<xfzkey.h>
#include<reg51.h>
#include<absacc.h>
#define CMD WR XBYTE[0x00]
#define DATA WR XBYTE[0x01]
#define BUSY RD XBYTE[0x02]
#define DATA RD XBYTE[0x03]
#define CLS 0x01
#define HOME 0x02
#define SETMODE 0x06
#define SETVISIBLE 0x08
#define SHIFT 0x10
#define SETFUNCTION 0x38
#define SETCGADDR 0x40
#define SETDDADDR 0x80
```

```

void busy(void);
void Delay(unsigned char j);
void lcdini(void)
{
    Delay(20);
    busy();
    CMD_WR=SETFUNCTION;
    busy();
    CMD_WR=SETVISIBLE;
    busy();
    CMD_WR=CLS;
    busy();
    CMD_WR=SETMODE;
    busy();
    CMD_WR=0x0c;
}
void main(void)
{
    lcdini();
    while(1)
    {
        busy();
        CMD_WR=0x80;
        busy();
        DATA_WR=led[key()];
    }
}
void busy(void)
{
    Delay(5);
}

xfzkey.h
#include <REG51.h>////////// MCS-51 头文件声明
#define uchar unsigned char//////////定义无符号变量简写式
#define uint unsigned int//////////
char led[]={ "0123456789ABCDEF" };
sfr KEY=0x90;//////////定义 P1 口为键盘接口
sfr Disp=0x80;
void Delay(uchar);//////////延时程序声明（单位为毫秒）

uchar key()//////////键盘处理函数
{
    uchar a,b,c;//////////定义 3 个变量
    KEY=0x0f;//////////键盘口置 00001111
    if (KEY!=0x0f)//////////查寻键盘口的值是否变化
    {
        Delay(20);//////////延时 20 毫秒
        if (KEY!=0x0f)//////////有键按下处理
        {

```

```

    a=KEY;//////////////////键值放入寄存器 a
}
KEY=0xf0;//////////////////将键盘口置为 11110000
c=KEY;//////////////////将第二次取得值放入寄存器 c
a=a|c;//////////////////将两个数据融合
switch(a)//////////////////对比数据值
{
    case 0xee: b = 0x00; break;///对比得到的键值给 b 一个应用数据
    case 0xde: b = 0x01; break;
    case 0xbe: b = 0x02; break;
    case 0x7e: b = 0x03; break;
    case 0xed: b = 0x04; break;
    case 0xdd: b = 0x05; break;
    case 0xbd: b = 0x06; break;
    case 0x7d: b = 0x07; break;
    case 0xeb: b = 0x08; break;
    case 0xdb: b = 0x09; break;
    case 0xbb: b = 0x0a; break;
    case 0x7b: b = 0x0b; break;
    case 0xe7: b = 0x0c; break;
    case 0xd7: b = 0x0d; break;
    case 0xb7: b = 0x0e; break;
    case 0x77: b = 0x0f; break;
    default:  break;///键值错误处理
}
}
return(b);///将 b 作为返回值
}

void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //1 毫秒延时
    }
}

```

实例七 用 1602LCD 显示 “HUANG HUAI UNIVERSITY”

1. 实例说明

要求在 1602LCD 的第 1 行第 1 列显示大写字母串 “HUANG HUAI”，第 2 行第 1 列显示 “UNIVERSITY”。16×2 显示、5×7 点阵、8 位数据口。显示可分以下步骤来完成。

- (1) LCD 初始化;
- (2) 写地址;
- (3) 写数据;
- (4) 显示字符串。

2. 仿真电路

用 1602LCD 显示 “HUANG HUAI UNIVERSITY” 仿真电路如图 6.18 所示。

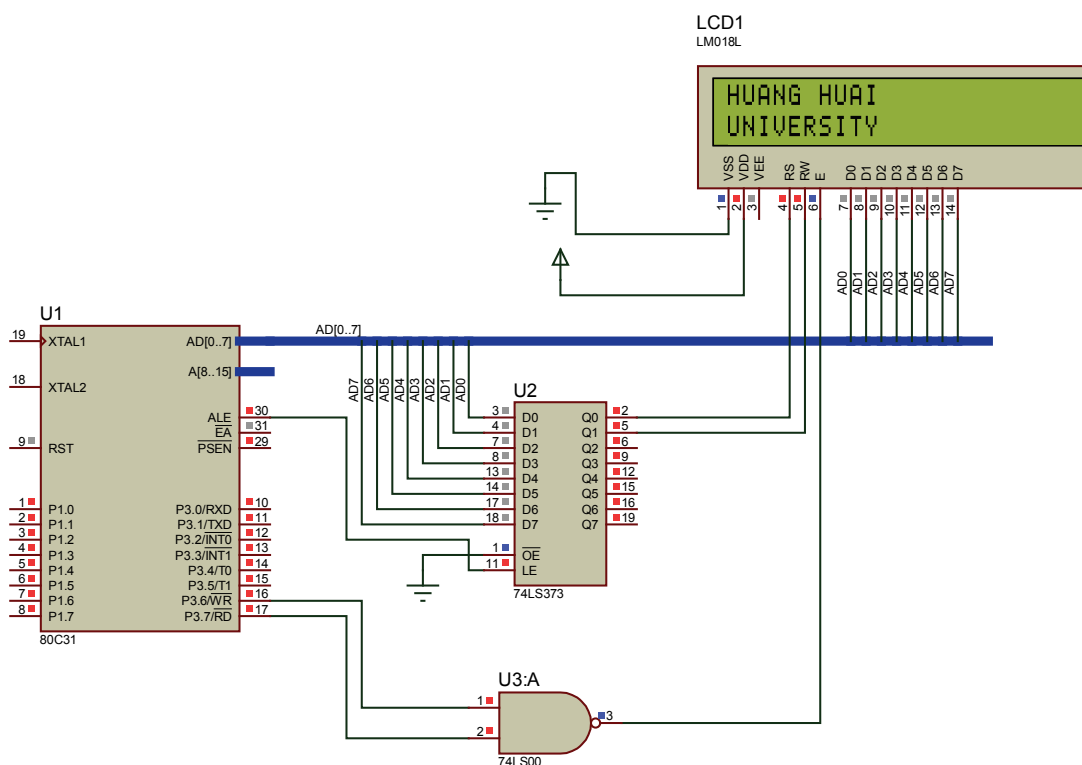


图 6.18 用 1602LCD 显示 “HUANG HUAI UNIVERSITY” 仿真电路

3. 程序设计

```
#include<reg51.h>
#include<absacc.h>
#define CMD_WR XBYTE[0x00]//命令寄存器
#define DATA_WR XBYTE[0x01]//数据寄存器写入
#define BUSY_RD XBYTE[0x02]//忙标志和地址计数器读出
#define DATA_RD XBYTE[0x03]// 数据寄存器读出
#define CLS 0x01//清屏
#define HOME 0x02 //返回
#define SETMODE 0x06//输入方式设置, I/D=1 增量方式, S=0 不移动
#define SETVISIBLE 0x08//显示开关控制, 关显示, 关光标, 字符不显示
#define SHIFT 0x10//光标移位, S/C=0 光标移位, R/L=0 向左移
#define SETFUNCTION 0x38//功能设置
#define SETCGADDR 0x40//CGRAM 地址设置
#define SETDDADDR 0x80//DDRAM 地址设置
char lcd[]={"HUANG HUAI UNIVERSITY"}; //要显示的字符串
void busy(void); //函数声明
void Delay(unsigned char j); //函数声明
```

```
void lcdini(void) //初始化函数
{
    Delay(20);
    busy();
    CMD_WR=SETFUNCTION; //功能设置
    busy();
    CMD_WR=SETVISIBLE; //显示关闭
    busy();
    CMD_WR=CLS; //清屏
    busy();
    CMD_WR=SETMODE; //显示光标移动设置
    busy();
    CMD_WR=0x0c; // 显示开及光标设置
}

void main(void)
{
    char i;
    lcdini(); //初始化
    while(1)
    {
        busy(); //查忙闲
        CMD_WR=0x80; //写行首地址
        for(i=0; i<11; i++)
        {
            busy(); //查忙闲
            DATA_WR=lcd[i]; //写数据
        }
        busy();
        CMD_WR=0xc0; //写行首地址
        for(i=11; i<22; i++)
        {
            busy();
            DATA_WR=lcd[i]; //写数据
        }
    }
}

void busy(void) //查忙闲函数
{
    Delay(5);
}

void Delay(unsigned char j) //延时
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++);
    }
}
```

4. 用开发板实验

本例程序编译通过后，生成 HEX 文件，烧录到 STC 单片机中，插入实验板，通电运行观察显示效果。

实例八 用 12864LCD 显示汉字

1. 实例说明

要求用 12864 显示汉字。

2. 仿真电路

用 12864LCD 显示汉字仿真电路如图 6.19 所示。

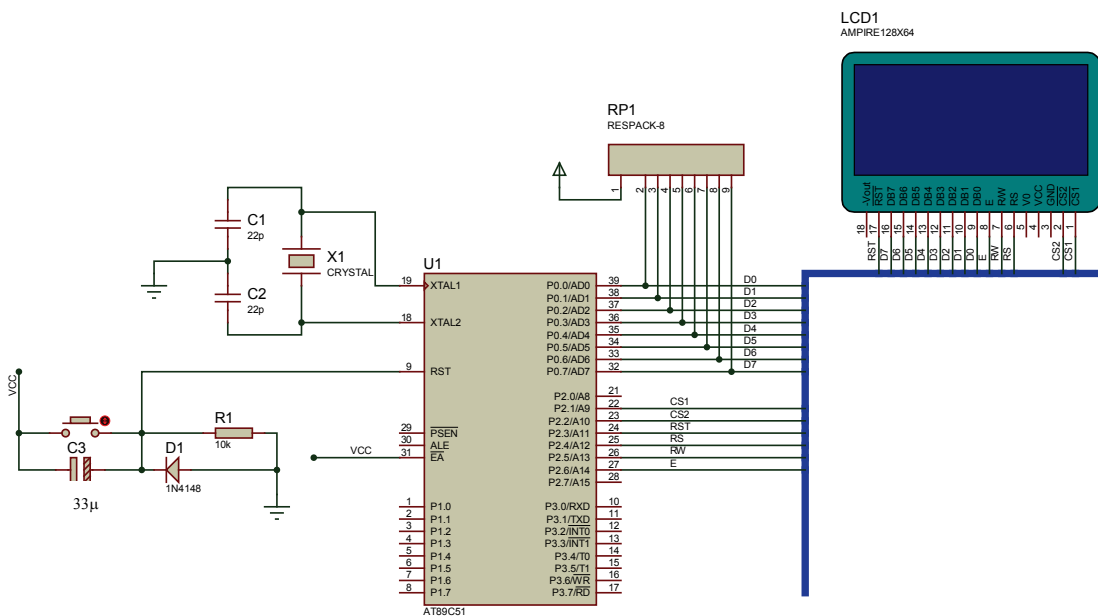


图 6.19 用 12864LCD 显示汉字仿真电路

3. 程序设计

```
#include <reg51.h>
#define LCDLCDDisp Off 0x3e
#define LCDLCDDisp On 0x3f
#define Page Add 0xb8//页地址
#define LCDCol Add 0x40//列地址
#define Start Line 0xc0//行地址
/*****液晶显示器的端口定义*****/
#define data ora P0 /*液晶数据总线*/
sbit LCDMcs=P2^1; /*片选 1*/
sbit LCDScs=P2^2; /*片选 2*/
sbit RESET=P2^3; /*复位信号*/
sbit LCDDi=P2^4; /*数据/指令 选择*/
```

```

sbit LCDRW=P2^5;      /*读/写 选择*/
sbit LCDEnable=P2^6;   /*读/写 使能*/
unsigned char code Bmp1[]={0x10,0x10,0x12,0xD2,0x52,0x5F,0x52,0xF2,0x52,
0x5F,0x52,0xD2,0x12,0x10,0x10,0x00,
0x00,0x00,0x00,0x9F,0x52,0x32,0x12,0x1F,0x12,0x32,0x52,0x9F,0x00,0x00,0x
00,0x00};
unsigned char code Bmp2[]={0x10,0x10,0x12,0xD2,0x52,0x5F,0x52,0xF2,0x52,
0x5F,0x52,0xD2,0x12,0x10,0x10,0x00,
0x00,0x00,0x00,0x9F,0x52,0x32,0x12,0x1F,0x12,0x32,0x52,0x9F,0x00,0x00,0x
00,0x00};
unsigned char code Bmp3[]={0x10,0x10,0x12,0xD2,0x52,0x5F,0x52,0xF2,0x52,
0x5F,0x52,0xD2,0x12,0x10,0x10,0x00,
0x00,0x00,0x00,0x9F,0x52,0x32,0x12,0x1F,0x12,0x32,0x52,0x9F,0x00,0x00,0x
00,0x00};
unsigned char code Bmp4[]={0x10,0x10,0x12,0xD2,0x52,0x5F,0x52,0xF2,0x52,
0x5F,0x52,0xD2,0x12,0x10,0x10,0x00,
0x00,0x00,0x00,0x9F,0x52,0x32,0x12,0x1F,0x12,0x32,0x52,0x9F,0x00,0x00,0x
00,0x00};
/** 函数功能:LCD 延时程序 入口参数:t 出口参数: **/
void LCDdelay(unsigned int t)
{
    unsigned int i,j;
    for(i=0;i<t;i++);
    for(j=0;j<10;j++);
}
/** 状态检查, LCD 是否忙**/
void CheckState()
{
    unsigned char dat,DATA;//状态信息(判断是否忙)
    LCDDi=0; // 数据\指令选择, D/I(RS)="L", 表示 DB7~DB0 为显示指令数据    LCDRW=1;
//R/W="H", E="H"数据被读到 DB7~DB0
    do
    {
        DATA=0x00;
        LCDEnable=1; //EN 下降源
        LCDdelay(2); //延时
        dat=DATA;
        LCDEnable=0;
        dat=0x80 & dat; //仅当第 7 位为 0 时才可操作(判别 busy 信号)
    }
    while(!(dat==0x00));
}
/** 函数功能:写命令到 LCD 程序, RS(DI)=L,RW=L,EN=H, 即来一个脉冲写一次 入口参
数:cmdcode 出口参数:**/
void write_com(unsigned char cmdcode)
{
    CheckState(); //检测 LCD 是否忙
    LCDDi=0;
    LCDRW=0;
    P0=cmdcode;

```

```

LCDdelay(2);
LCDEnable=1;
LCDdelay(2);
LCDEnable=0;
}
/** 函数功能:LCD 初始化程序 入口参数: 出口参数:**/
void init_lcd()
{
LCDdelay(100);
LCDMcs=1;//刚开始关闭两屏
LCDScs=1; LCDdelay(100);
write_com(LCDLCDDisp_Off); //写初始化命令
write_com(Page_Add+0);
write_com(Start_Line+0);
write_com(LCDCol_Add+0);
write_com(LCDLCDDisp_On);
}
/** 函数功能:写数据到 LCD 程序, RS(DI)=H, RW=L, EN=H, 即来一个脉冲写一次 入口参
数:LCDDispdata 出口参数: **/
void write_data(unsigned char LCDDispdata)
{
CheckState();//检测 LCD 是否忙
LCDDi=1;
LCDRW=0;
P0=LCDDispdata;
LCDdelay(2);
LCDEnable=1;
LCDdelay(2);
LCDEnable=0; }
/** 函数功能:清除 LCD 内存程序 入口参数:pag,col,hzk 出口参数:***/
void Clr_Scr()
{
unsigned char j,k;
LCDMcs=0; //左、右屏均开显示
LCDScs=0;
write_com(Page_Add+0);
write_com(LCDCol_Add+0);
for(k=0;k<8;k++)//控制页数 0~7, 共 8 页
{
write_com(Page_Add+k); //每页每页进行写
for(j=0;j<64;j++) //每页最多可写 32 个中文字或 64 个 ASCII 字符
{
write_com(LCDCol_Add+j);
write_data(0x00);//控制列数 0-63, 共 64 列, 写点内容, 列地址自动加 1
}
}
}
/** 函数功能:左屏位置显示 入口参数:page,column,hzk 出口参数:**/
void Bmp_Left_Disp(unsigned char page,unsigned char column, unsigned char
code *Bmp)

```



```

{
    unsigned char j=0,i=0;
    for(j=0;j<2;j++)
    {
        write_com(Page_Add+page+j);
        write_com(LCDCol_Add+column);
        for(i=0;i<64;i++)
            write_data(Bmp[128*j+i]);
    }
}

/** 函数功能:右屏位置显示 入口参数:page,column,hzk 出口参数:**/
void Bmp_Right_Disp(unsigned char page,unsigned char column, unsigned char
code *Bmp)
{
    unsigned char j=0,i=0;
    for(j=0;j<2;j++)
    {
        write_com(Page_Add+page+j);
        write_com(LCDCol_Add+column);
        for(i=64;i<128;i++)
            write_data(Bmp[128*j+i]);
    }
}

void main()
{
    init_lcd();
    Clr_Scr();
    LCDMcs=0; //左屏开显示
    LCDScs=1;
    Bmp_Left_Disp(0,0,Bmp1); // Bmp1 为某个汉字的首地址
    Bmp_Left_Disp(2,0,Bmp2);
    Bmp_Left_Disp(4,0,Bmp3);
    Bmp_Left_Disp(6,0,Bmp4);
    LCDMcs=1; //右屏开显示
    LCDScs=0;
    Bmp_Right_Disp(0,0,Bmp1);
    Bmp_Right_Disp(2,0,Bmp2);
    Bmp_Right_Disp(4,0,Bmp3);
    Bmp_Right_Disp(6,0,Bmp4);
    while(1)
    {
    }
}

```

本章小结

本章主要介绍了键盘、7段显示器和 LCD1602 的扫描、显示原理，并给出了具体的应用实例，以便于读者学习键盘、显示接口设计。

习题六

一、简答题

1. 简述 7 段显示器的动态显示过程。
2. 简述矩阵式键盘的显示过程。
3. 简述按键去抖动的方法。
4. 简述 LCD1602 的初始化过程。

二、设计编程题

1. 某控制系统有 2 个开关 K1 和 K2, 1 个数码管, 当 K1 按下时数码管加 1, K2 按下时数码管减 1。试画出 8051 与外设的连接图并编程实现上述要求。
2. 据行列式键盘线反转法识别按键的原理, 编写识别 0~3 按键的扫描程序。
3. 利用单片机的串行口和 74LS164 扩展一显示接口, 在 7 段显示器上循环显示 0~9 的数字。
 - (1) 完成电路设计;
 - (2) 编写程序。
4. 通过两个按键对变量 x 进行加 1 和减 1 操作, 并把 x 的值送 LCD1602 显示。



第7章 单片机应用系统设计与调试简介

学习目标

了解单片机应用系统的开发和调试过程及抗干扰设计技术。

重点难点

单片机应用系统的开发设计方法和可靠性设计。

7.1 知识结构

单片机应用系统是指以单片机为核心，配以外围电路和软件，能完成某种或几种功能的应用系统。因此，单片机应用系统的设计包括硬件设计、软件设计、电路板制作、焊接组装与系统调试。

为了保证系统能可靠工作，在软、硬件的设计中，还要考虑其抗干扰设计。在应用系统的设计中，软件、硬件和抗干扰设计是紧密相关、不可分离的。在有些情况下硬件的任务可由软件来完成（如某些滤波、校准功能等）；而在另一些要求系统实时性强、响应速度快的场合，则往往用硬件来替代软件来完成某些功能。设计者应根据实际情况，合理地安排软、硬件的比例，选取最佳的设计方案，使系统具有最佳的性能价格比。

7.1.1 单片机应用系统的设计步骤

单片机应用系统包括硬件和软件两部分，硬件是系统的基础，软件则是在硬件的基础上完成特殊的任务。单片机应用系统的开发过程往往是硬件与软件协同开发的过程。

应用系统开发主要可以分为如下几个主要阶段。

（1）需求分析与方案论证和总体设计方案

主要包括需求分析、可行性调研、技术指标的确定、器件的选择和软件功能的划分等。

（2）应用系统硬件、软件和抗干扰设计

主要包括硬件、软件设计和抗干扰设计。其中硬件主要包括键盘、显示、A/D 电路等外围扩展电路的设计和地址译码、总线驱动等电路设计。软件设计则主要包括定义系统功能、画出程序流程图和编写代码等。作为实际的产品，除了满足基本的功能外，还必须考虑可靠性设计的问题。

（3）系统组装与调试

主要包括硬件调试、软件调试及软硬件联合调试。硬件调试主要包括静态调试和动态调试。软件调试主要是在线的仿真调试。调试中一般软件和硬件不可能完全分开，软件调

试和硬件调试通常要协同完成。

(4) 固化应用程序、试运行

完成系统调试之后,反复运行正常则可将用户系统程序固化到 EPROM 之类的存储器上,单片机脱离开发系统独立工作,并在试运行阶段观测所设计系统是否满足设计要求。

(5) 资料与文件整理编制阶段

系统调试通过,就进入资料与文件整理编制阶段。资料与文件包括:任务描述、设计的指导思想及设计方案论证、性能测定及现场试用报告与说明、使用指南、软件资料(流程图、子程序使用说明、地址分配、程序清单)、硬件资料(电原理图、元件布置图及接线图、插接件引脚图、线路板图、注意事项)。文件不仅是设计工作的结果,而且是以后使用、维修以及进一步再设计的依据。因此,要精心编写,描述清楚,使数据及资料齐全。

7.1.2 应用系统的硬件设计

硬件设计的主要任务是根据总体设计要求,在所选器件的基础上,确定系统扩展所要用的存储器、I/O 电路、A/D 及有关外围电路等。另外,为了使系统稳定可靠地工作,在满足功能之余,还必须进行硬件的可靠性设计。

1. 硬件设计的具体步骤

(1) 绘制硬件方框图:根据给定的总体任务,确定数字电路和模拟电路所需要的模块,画出总体的硬件方框图,确定硬件的总体方案。

(2) 确定数据输入输出的方式:确定各输入输出数据的传送方式是中断方式、查询方式还是无条件方式等。

(3) 硬件资源分配:各输入输出信号分别使用哪个并行口、串行口、中断、定时器/计数器等。

(4) 绘制原理图:根据以上各步的分析结果完成硬件的电气连接原理图。

(5) 制作电路板:根据绘制的电路原理图,绘制出 PCB 图,并送厂家生产,得到实际的电路板。

(6) 器件焊接:将所有的元器件焊接到制作的电路板上。

2. 在单片机应用系统硬件设计中应注意的事项

(1) 尽可能选择典型电路,并符合单片机常规用法,为硬件系统的标准化、模块化打下良好的基础。

(2) 系统扩展与外围设备的配置水平应充分满足应用系统的功能要求,并留有适当余地,以便进行二次开发。

(3) 硬件结构应结合应用软件方案一并考虑。硬件结构与软件方案会产生相互影响,考虑原则是:软件能实现的功能尽可能由软件实现,以简化硬件结构。但必须注意,由软件实现的硬件功能,一般响应时间比硬件实现长,且占用 CPU 时间。

(4) 系统中的相关器件要尽可能做到性能匹配。如选用 CMOS 芯片单片机构成低功耗系统时,系统中所有芯片都应尽可能选择低功耗产品。如果既有 CMOS 电路,又有 TTL 电路时,要设计相应的电平兼容和转换电路。当有 RS-232 和 RS-485 接口时,还要实现电平兼容和转换。常用的集成电路有 MAX232 和 MAX485 等。

(5) 要考虑负载容限问题。单片机总线的负载能力是有限的。如 MCS-51 的 P0 口的负载能力为 4 mA, 最多驱动 8 个 TTL 电路, P1~P3 口的负载能力为 2 mA, 最多驱动 4 个 TTL 电路。若外接负载较多, 则应采取总线驱动的方法提高系统的负载容限。常用驱动器有: 单向驱动器 74LS244, 双向驱动器 74LS245 等。

(6) 尽量往“单片”方向设计硬件系统。系统器件越多, 器件之间相互干扰也越强, 功耗也相应增大, 也不可避免地降低了系统的稳定性。随着单片机片内集成的功能越来越强, 在设计中尽量选择性能更优、功能更强的芯片。

7.1.3 应用系统的软件设计

一般来说, 单片机中的软件功能可分为两大类: 一类是执行软件, 它完成各种实质性的功能, 如测量、计算、显示、打印、输出控制等; 另一类是监控软件, 它专门用来协调各执行模块和操作者之间的关系, 充当组织调度的角色。

在软件设计中, 还应注意如下事项。

(1) 根据软件功能要求, 将系统软件分成若干个相对独立的部分。根据它们之间的联系和时间上的关系, 设计出合理的软件总体结构, 使其清晰、简洁、流程合理。

(2) 培养结构化程序设计风格, 各功能程序实行模块化、自程序化。既便于调试、连接, 又便于移植、修改。

(3) 为提高软件设计的总体效率, 以简明、直观的方法对任务进行描述, 在编写应用软件前, 应绘制出程序流程图。这不仅是程序设计的一个重要组成部分, 而且是决定成败的关键部分。从某种意义上讲, 多花一份时间来设计程序流程图, 就可以节约几倍源程序编辑调试的时间。

(4) 要合理分配系统资源, 包括 ROM、RAM、定时器/计数器、中断源等。其中最关键的是片内 RAM 分配。当各种资源规划好后, 应列出一张资源详细分配表, 以备编程查用。

(5) 注意在程序的有关位置写上功能注释, 提高程序的可读性。

7.1.4 单片机应用系统的开发与调试

在完成目标系统样机的组装和软件设计之后, 便进入系统的调试阶段。用户系统的调试步骤和方法是相同的, 但具体细节则与所采用的开发系统以及目标系统所选用的单片机型号有关。系统调试的目的是查出系统中硬件设计与软件设计中存在的错误及可能出现的不协调的问题, 以便修改设计, 最终使系统能正确地工作。系统调试包括硬件调试、软件调试和软、硬件联调。

1. 硬件调试

当硬件设计从布线到焊接安装完成之后, 就开始进入硬件调试阶段。硬件调试的常用工具包括仿真器、万用表、逻辑笔、函数信号发生器、逻辑分析仪、示波器等。硬件调试可按静态调试和动态调试两步进行。

(1) 静态调试

静态调试是指在系统加电前的检查, 主要是排除明显的硬件故障。静态调试的内容包

括以下 3 个方面。

① 排除逻辑故障：这类故障往往是由于设计和加工印制板过程中工艺性错误所造成的。主要包括错线、开路、短路。排除的方法是首先将加工的印制板认真对照原理图，看两者是否一致。应特别注意电源系统检查，以防止电源短路和极性错误，并重点检查系统总线（地址总线、数据总线和控制总线）是否存在相互之间短路或与其他信号线路短路。必要时利用数字万用表的短路测试功能，可以缩短排错时间。

② 排除元器件失效：造成这类错误的原因有两个，一是元器件买来时就已坏了；另一个是由于安装错误，造成器件烧坏。可以采取检查元器件与设计要求的型号、规格和安装是否一致。在保证安装无误后，用替换方法排除错误。

③ 排除电源故障：在通电前，一定要检查电源电压的幅值和极性，否则很容易造成集成块损坏。加电后检查各插件引脚上的电位，一般先检查 V_{CC} 与 GND 之间电位，若在 5~8 V 之间属正常。若有高压，有时会使用应用系统中的集成块发热损坏。

（2）动态调试

动态调试又称为联机调试，主要是在静态调试的基础上排除硬件系统中存在的其他问题。联机前先断电，将单片机仿真器的仿真头插到样机的单片机插座上，检查一下仿真器与样机之间的电源、接地是否良好。

通电后执行开发机的读写指令，对用户样机的存储器、I/O 端口进行读写操作、逻辑检查，若有故障，可用示波器观察有关波形（如选中的译码器输出波形、读写控制信号、地址数据波形以及有关控制电平）。通过对波形的观察分析，寻找故障原因，并进一步排除故障。可能的故障有：线路连接上有逻辑错误、有开路或短路现象、集成电路失效等。

在用户系统的样机（主机部分）调试好后。可以插上用户系统的其他外围部件如键盘、显示器、输出驱动板、A/D、D/A 板等，再将这些电路进行初步调试。

在调试过程中若发现用户系统工作不稳定，可能存在下列情况：

电源系统供电电流不足，联机时公共地线接触不良；用户系统主板负载过大；用户的各级电源滤波不完善等。对这些问题一定要认真查出原因，加以排除。

2. 软件调试

软件调试与所选用的软件结构和程序设计技术有关。如果采用模块程序设计技术，则逐个模块分别调试。调试各子程序时一定要符合现场环境，即入口条件和出口条件。调试的手段可采用单步或设断点运行方式，通过检查用户系统 CPU 的现场、RAM 的内容和 I/O 口的状态，检查程序执行结果是否符合设计要求。通过检测可以发现程序中的死循环错误、机器码错误及转移地址的错误，同时也可以发现用户系统中的硬件故障、软件算法及硬件设计错误。在调试过程中不断调整用户系统的软件和硬件，逐步通过一个一个程序模块。

各模块通过以后，可以把有关的功能块联合起来一起进行综合调试。在这个阶段若发生故障，可以考虑各子程序在运行时是否破坏现场，缓冲单元是否发生冲突，标志位的建立和清除在设计上有没有失误，堆栈区域有无溢出，输入设备的状态是否正常，等等。若用户系统是在开发机的监控程序下运行时，还要考虑用户缓冲单元是否和监控程序的工作单元发生冲突。

单步和断点调试后，还应进行连续调试，这是因为单步运行只能验证程序的正确与否，

而不能确定定时精度、CPU 的实时响应等问题。待全部调试完成后，应反复运行多次，除了观察稳定性之外，还要观察用户系统的操作是否符合原始设计要求、安排的用户操作是否合理等，必要时再作适当修正。

如果采用实时多任务操作系统，一般是逐个任务进行调试。调试方法与上基本相似，只是实时多任务操作系统的应用程序是由若干个任务程序组成的，一般是逐个任务进行调试，在调试某一任务时，同时也调试相关的子程序、中断服务程序和一些操作系统的程序。调试好以后，再使各个任务程序同时运行，如果操作系统无错误，一般情况下系统就能正常运转。

3. 系统联调

硬件和软件经调试完成后，对用户系统要进行现场实验运行，检查软硬件是否按预期的要求工作，各项技术指标是否达到设计要求。一般而言，系统经过软硬件调试之后均可以正常工作。但在某些情况下，由于单片机应用系统运行的环境较为复杂，尤其在干扰较严重的场合下，在系统进行实际运行之前无法预料，只能通过现场运行来发现问题，以找出相应的解决办法。或者虽然已经在系统设计时采取了软硬件抗干扰措施，但效果如何，还需通过在现场运行才能得到验证。

7.1.5 单片机应用系统的可靠性与抗干扰性设计

鉴于单片机主要应用在工业控制、智能仪表、家用电器等领域，因此对基于单片机的应用系统的可靠性、抗干扰性、自诊断等提出了较高要求。

7.1.5.1 系统的可靠性设计

所谓可靠性，有广义和狭义两种解释。广义是指产品在其整个生命周期内完成规定功能的能力，它包括狭义可靠性和维修性。这里讲的狭义可靠性是指产品在规定的条件和时间内，完成规定功能的能力。也就是说，在规定的时间内完成规定功能的可能性或概率。

提高系统的可靠性也就是减小系统的故障率，一般引起系统故障的原因有以下两个方面。

- (1) 外部因素：例如环境温度、湿度、电源电压、电磁干扰、冲击、化学腐蚀等。
- (2) 内部因素：包括软件和硬件两个部分。

下面主要就内部因素中的硬件和软件的可靠性设计进行介绍。

1. 硬件可靠性设计

满足基本功能的单片机应用系统，是否能真正应用于实践，还必须考虑到可靠性的问题。影响单片机系统可靠安全运行的主要因素主要来自系统内部和外部的各种电气干扰，并受系统结构设计，元器件选择、安装、制造工艺影响。如果不充分考虑到这些干扰因素，并采取相应的抗干扰措施，常会导致单片机应用系统运行失常，轻则影响产品质量，限制其使用范围，重则会导致单片机应用系统根本不能应用于实际。形成干扰的基本要素有 3 个方面。

(1) 干扰源：指产生干扰的元件、设备或信号，如雷电、继电器、可控硅、电机、高频时钟等都可能成为干扰源。

(2) 传播路径：指干扰从干扰源传播到敏感器件的通路或媒介，典型的干扰传播路

径是通过导线的传导和空间的辐射。

(3) 敏感器件：指容易被干扰的对象，如 A/D、D/AI 转换器，单片机，数字 IC，弱信号放大器等。

针对形成干扰的 3 个要素，硬件抗干扰经常采取的措施主要有以下两种。

(1) 抑制干扰源

抑制干扰源是抗干扰设计中最优先考虑和最重要的原则，抑制干扰源的常用措施如下。

① 继电器线圈增加续流二极管，消除断开线圈时产生的反电动势干扰。仅加续流二极管会使继电器的断开时间滞后，增加稳压二极管后继电器在单位时间内可动作更多的次数。

② 在继电器接点两端并接火花抑制电路（一般是 RC 串联电路，电阻器一般选几千欧姆到几万欧姆，电容器选 $0.01\ \mu\text{F}$ ），减小电火花影响。

③ 给电动机加滤波电路，注意电容器、电感器引线要尽量短。

④ 电路板上每个集成电路要并接一个 $0.01\sim 0.1\ \mu\text{F}$ 的高频电容器，以减小集成电路对电源的影响。注意高频电容器的布线，连线应靠近电源端并尽量粗短，否则，等于增大了电容量的等效串联电阻值，会影响滤波效果。

⑤ 布线时避免 90° 折线，减少高频噪声发射。

⑥ 可控硅两端并接 RC 抑制电路，减小可控硅产生的噪声（这个噪声严重时可能会把可控硅击穿）。

(2) 切断干扰传播路径及抗干扰措施

按干扰的传播路径可分为传导干扰和辐射干扰两类。所谓传导干扰是指通过导线传播到敏感器件的干扰。高频干扰噪声和有用信号的频带不同，可以通过在导线上增加滤波器的方法切断高频干扰噪声的传播，有时也可加隔离光电耦合器来解决。电源噪声的危害最大，要特别注意处理。所谓辐射干扰是指通过空间辐射传播到敏感器件的干扰。一般的解决方法是增加干扰源与敏感器件的距离，用地线把它们隔离并在敏感器件上加屏蔽罩。切断干扰传播路径的常用措施如下。

① 充分考虑电源对单片机的影响。电源做得好，整个电路的抗干扰就解决了一大半。许多单片机对电源噪声很敏感，要给单片机电源加滤波电路或稳压器，以减小电源噪声对单片机的干扰。比如，可以利用磁珠和电容器组成“ Π ”形滤波电路，当然条件要求不高时也可用 $100\ \Omega$ 电阻器代替磁珠。

② 如果单片机的 I/O 口用来控制电动机等噪声器件，在 I/O 口与噪声源之间应加隔离（增加 Π 形滤波电路）。

③ 注意晶振布线。晶振与单片机引脚尽量靠近，用地线把时钟区隔离起来，晶振外壳接地并固定。

④ 电路板合理分区，如强、弱信号，数字、模拟信号。尽可能把干扰源（如电动机、继电器）与敏感元件（如单片机）远离。

⑤ 用地线把数字区与模拟区隔离。数字地与模拟地要分离，最后在一点接于电源地。A/D、D/A 芯片布线也以此为原则。

⑥ 单片机和大功率器件的地线要单独接地，以减小相互干扰。大功率器件尽可能放在电路板边缘。

⑦ 在单片机 I/O 口、电源线、电路板连接线等关键地方使用抗干扰元件，如磁珠、磁环、电源滤波器、屏蔽罩，可显著提高电路的抗干扰性能。

提高敏感器件的抗干扰性能是指从敏感器件这边考虑尽量减少对干扰噪声的拾取，以及从不正常状态尽快恢复的方法。提高敏感器件抗干扰性能的常用措施如下。

- 布线时尽量减小回路环的面积，以降低感应噪声。
- 布线时，电源线和地线要尽量粗。除减低压降外，更重要的是降低耦合噪声。
- 对于单片机闲置的 I/O 口，不要悬空，要接地或接电源。其他 IC 的闲置端在不改变系统逻辑的情况下接地或接电源。
- 对单片机使用电源监控及看门狗电路，如：IMP809、IMP706、IMP813、X5043、X5045 等，可大幅度提高整个电路的抗干扰性能。
- 在速度能满足要求的前提下，尽量降低单片机的晶振和选用低速数字电路。
- 集成电路器件尽量直接焊在电路板上，少用集成电路插座。

⑧ 其他常用抗干扰措施有以下几种。

- 交流端用电感电容器滤波器：去掉高频低频干扰脉冲。
- 变压器双隔离措施：变压器初级输入端串接电容器，初、次级线圈间屏蔽层与初级间电容器中心接点接大地，次级外屏蔽层接印制板地，这是硬件抗干扰的关键手段。次级加低通滤波器吸收变压器产生的浪涌电压。
- 采用集成式直流稳压电源：因为有过流、过压、过热等保护。
- I/O 口采用光电、磁电、继电器隔离，同时去掉公共地。
- 通信线用双绞线以排除平行互感。
- 防雷电用光纤隔离最为有效。
- A/D 转换用隔离放大器或采用现场转换以减小误差。
- 外壳接大地以解决人身安全及防外界电磁场干扰。
- 加复位电压检测电路以防止复位不充分 CPU 就工作。
- 有条件采用四层以上印制板，中间两层为电源及接地。

2. 软件可靠性设计

单片机应用系统用于实际工作中时，如果硬件出现干扰，可能会导致程序运行混乱。因此单片机应用系统的软件应在完成基本功能之外，必须保证程序运行混乱后还能重新进入正轨。在单片机应用系统设计中，在提高硬件系统抗干扰能力的同时，还要进行软件抗干扰的设计。常用的软件抗干扰方法主要有以下几种。

(1) 指令冗余

CPU 取指令过程是先取操作码，再取操作数。当 PC 受干扰出现错误，程序便脱离正常轨道“乱飞”，当“乱飞”到某双字节指令，若取指令时落在操作数上，误将操作数当作操作码，程序将出错。若“飞”到了三字节指令，出错机率更大。在关键地方人为插入一些单字节指令，或将有效单字节指令重写，这称为指令冗余。通常是在双字节指令和三字节指令后插入两个字节以上的 NOP。这样即使乱飞程序飞到操作数上，由于空操作指令 NOP 的存在，避免了后面的指令被当作操作数执行，程序自动纳入正轨。此外，对系统流向起重要作用的指令如 RET、RETI、LCALL、LJMP、JC 等指令之前插入两条 NOP，也

可将乱飞程序纳入正轨，确保这些重要指令的执行。

(2) 拦截技术

当乱飞程序进入非程序区，冗余指令便无法起作用，这时可以用拦截技术，将程序引向指定位置，再进行出错处理。通常用软件陷阱来拦截乱飞的程序。软件陷阱是指用来将捕获的乱飞程序引向复位入口地址 0000H 的指令。通常在 EPROM 中非程序区填入以下指令作为软件陷阱：

```
NOP;
NOP;
LJMP0000H;
```

当乱飞程序落到此区，即可自动入轨。在用户程序区各模块之间的空余单元也可填入陷阱指令。当使用的中断因干扰而开放时，在对应的中断服务程序中设置软件陷阱，能及时捕获错误的中断。如某应用系统虽未用到外部中断 1，外部中断 1 的中断服务程序可为如下形式：

```
NOP;
NOP;
RETI;
```

返回指令可用“RETI”，也可用“LJMP 0000H”。

(3) 软件“看门狗”技术

若失控的程序进入“死循环”，通常采用“看门狗”技术使程序脱离“死循环”。主程序不断检测程序循环运行时间，若发现程序循环时间超过最大循环运行时间，则认为系统陷入“死循环”，需进行出错处理。

“看门狗”技术可由硬件实现，也可由软件实现。在工业应用中，严重的干扰有时会破坏中断方式控制字，关闭中断。则系统无法定时“喂狗”，硬件看门狗电路失效。而软件看门狗可有效地解决这类问题。

7.1.5.2 系统自诊断

自诊断又称“自检”，是通过软硬件配合来实现对系统故障的自动检测，一般有上电自检、定时自检、键控自检三种形式。通过自检可以及时发现系统问题，防止程序出错，从而增强系统运行的可靠性。系统的自检一般包括以下 5 个部分。

1. CPU 的自检

CPU 的自检包括指令系统的诊断、片内 RAM 诊断、定时器和中断的诊断等。单片机执行完一个包含有传送指令、算术运算指令、逻辑运算指令、位传送指令、位逻辑操作指令的程序以后，累加器 A 中的数据应该为预定值，否则出错。对于片内 RAM 的诊断可采用如下过程对每一个单元进行测试：读出→备份→写入→再读出→与备份比较，若相同则重新写入原单元，否则应设置不正确标志位标明片内 RAM 有问题。对于定时器及中断的诊断，一般采用以下方法：

用软件延时来检测定时器的准确性，即让定时器工作在定时方式，如果能按时输出，则置溢出标志位为“1”，否则表明定时器有问题。利用定时中断来检测中断系统是否有问

题,即若允许定时中断,并在中断服务程序中做一件事通知自检程序,则可以根据这件事是否发生来判断中断是否正常。

2. ROM 的诊断

ROM 的诊断通常采用静态测试法。在将系统程序和自检程序固化到 ROM 之前,先要计算其机器代码的累加和,并取其结果的低 16 位,将这个累加和同结果一起固化到 ROM 特定的单元中。在对 ROM 进行检查时,只需对固化在 ROM 的程序代码计算累加并将结果和事先计算好的那个累加和进行比较,若相等则说明 ROM 完好。

3. 外部 RAM 的诊断

对于外部 RAM 的诊断可以采用与 ROM 诊断类似的方法进行。另外,由于 RAM 的故障太多,是以大片存储区域被破坏的形式出现的,因此可以采用 RAM 分段放置标志位的方法来判断 RAM 是否被破坏。

4. A/D、D/A 转换通道的诊断

一般的单片机系统都带有模拟量的采样电路。在模拟量不多的情况下,一般用一个转换芯片就可以完成采样。若 A/D 转换芯片自身不带多路转换开关,则还要采用多路模拟开关来切换各路输入信号,实现分时采样转换。A/D 转换通道的诊断方法是在某一路模拟量的输入端加上一个已知的模拟电压,启动 A/D 转换读出转换结果。如果结果在预定值允许的误差范围内,则说明转换芯片工作正常。当转换结果和预定值有较大误差时,应该检查电路 A/D 芯片的工作电压及其他元器件的参数是否匹配,同时还可通过软件手段进行校正。

D/A 通道诊断需要借助 A/D 的一个输入通道,在已经进行过 A/D 诊断并确定该通道正常以后,将预定值送给 D/A 转换。转换以后的模拟电压通过分压电阻器接到 A/D 转换的某个输入端,启动 A/D 转换得到变化以后的数字量。将 D/A 送出的数字量和读入的数字量进行比较,若在允许的误差范围内则说明 D/A 转换通道工作正常。

5. I/O 通道的诊断

单片机系统的 I/O 通道在很多情况下都用作数字显示和键盘接口等。数字显示功能通常要用到数码管,显示的内容通常有数字、小数点、符号、提示符等。在编写自检程序时可将数码管的所有段位点亮,检查数码管是否缺段。也可以设置循环输出全 0 到全 9 的数字、小数点在各位上循环显示,以及显示特定的提示信息,从而检查各相关的 I/O 通道是否正常工作。对于键盘的诊断可采用以下方法:当按下某个键后,自检程序可通过一个 I/O 口驱动蜂鸣器发声或驱动一个数码管显示特定的符号,如果发声或者显示正常则说明该键正常;如果是单个键有问题,一般是接触不良引起的;如果是整排键没响应,则是键扫描电路出了故障。

以上介绍了单片机系统中几个主要硬件电路部分的自诊断常用方法,供读者参考使用。当然,在实际的应用中也可以根据系统的需要来选择若干个项目组合成一个完整的硬件自诊断系统。

7.2 学习实例

实例一 基于 DS1302 的日历时钟设计

1. 实例说明

要求用单片机、DS1302、LCD1602 设计一个日历时钟。请查阅 DS1302 资料。

2. 仿真电路

基于 DS1302 的日历时钟设计仿真电路如图 7.1 所示。

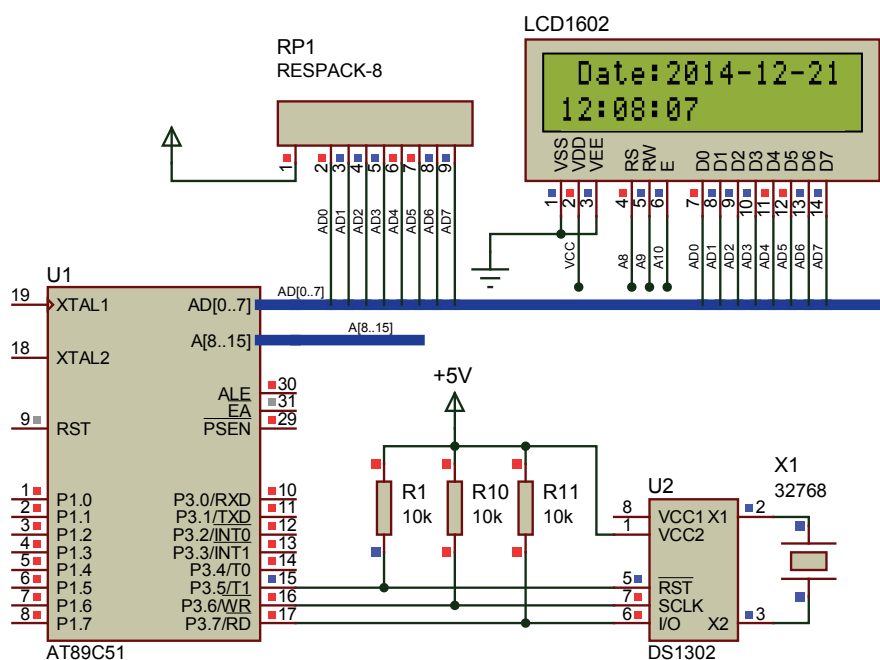


图 7.1 基于 DS1302 的日历时钟设计仿真电路

3. 程序设计

```
#include<reg51.h>    //包含单片机寄存器的头文件
#include<intrins.h>  //包含_nop_()函数定义的头文件
#include<stdio.h>
/*****
以下是 DS1302 芯片的操作程序
*****/

/
unsigned char code digit[10]={"0123456789"}; //定义字符数组显示数字
sbit RST=P3^5;    //位定义 1302 芯片的接口, 复位端口定义在 P3.5 引脚
sbit SCLK=P3^6;   //位定义 1302 芯片的接口, 时钟输出端口定义在 P3.6 引脚
sbit DATA=P3^7;  //位定义 1302 芯片的接口, 数据输出端口定义在 P3.7 引脚
```

```

bit ReadTempFlag;//定义读时间标志
/*****
函数功能: 延时若干微秒
入口参数: n
*****/
void delaynus(unsigned char n)
{
    unsigned char i;
    for(i=0;i<n;i++)
        ;
}
/*****
函数功能: 向 1302 写一个字节数据
入口参数: x
*****/
void Write1302(unsigned char dat)
{
    unsigned char i;
    SCLK=0;           //拉低 SCLK, 为脉冲上升沿写入数据做好准备
    delaynus(2);       //稍微等待, 使硬件做好准备
    for(i=0;i<8;i++)   //连续写 8 个二进制位数据
    {
        DATA=dat&0x01; //取出 dat 的第 0 位数据写入 1302
        delaynus(2);     //稍微等待, 使硬件做好准备
        SCLK=1;          //上升沿写入数据
        delaynus(2);     //稍微等待, 使硬件做好准备
        SCLK=0;          //重新拉低 SCLK, 形成脉冲
        dat>>=1;         //将 dat 的各数据位右移 1 位, 准备写入下一个数据位
    }

}
/*****
函数功能: 根据命令字, 向 1302 写一个字节数据
入口参数: Cmd, 储存命令字; dat, 储存待写的数据
*****/
void WriteSet1302(unsigned char Cmd,unsigned char dat)
{
    RST=0;           //禁止数据传递
    SCLK=0;          //确保写数居前 SCLK 被拉低
    RST=1;           //启动数据传输
    delaynus(2);     //稍微等待, 使硬件做好准备
    Write1302(Cmd);  //写入命令字
    Write1302(dat);  //写数据
    SCLK=1;          //将时钟电平置于已知状态
    RST=0;           //禁止数据传递
}
/*****
函数功能: 从 1302 读一个字节数据
入口参数: x
*****/

```

```

unsigned char Read1302(void)
{
    unsigned char i,dat;
    delaynus(2);          //稍微等待，使硬件做好准备
    for(i=0;i<8;i++)      //连续读 8 个二进制位数据
    {
        dat>>=1;          //将 dat 的各数据位右移 1 位，因为先读出的是字节的最低位
        if(DATA==1)        //如果读出的数据是 1
            dat|=0x80;      //将 1 取出，写在 dat 的最高位
        SCLK=1;            //将 SCLK 置于高电平，为下降沿读出
        delaynus(2);        //稍微等待
        SCLK=0;            //拉低 SCLK，形成脉冲下降沿
        delaynus(2);        //稍微等待
    }
    return dat;            //将读出的数据返回
}
/*****
函数功能：根据命令字，从 1302 读取一个字节数据
入口参数：Cmd
*****/
unsigned char ReadSet1302(unsigned char Cmd)
{
    unsigned char dat;
    RST=0;                //拉低 RST
    SCLK=0;                //确保写数居前 SCLK 被拉低
    RST=1;                //启动数据传输
    Write1302(Cmd);        //写入命令字
    dat=Read1302();        //读出数据
    SCLK=1;                //将时钟电平置于已知状态
    RST=0;                //禁止数据传递
    return dat;            //将读出的数据返回
}
/*****
函数功能：1302 进行初始化设置
*****/
void Init_DS1302(void)
{
    WriteSet1302(0x8E,0x00); //根据写状态寄存器命令字，写入不保护指令
    WriteSet1302(0x80,((0/10)<<4|(0%10))); //根据写秒寄存器命令字，写入秒的初始值
    WriteSet1302(0x82,((0/10)<<4|(8%10))); //根据写分寄存器命令字，写入分的初始值
    WriteSet1302(0x84,((12/10)<<4|(2%10))); //根据写小时寄存器命令字，写入小时的
初始值
    WriteSet1302(0x86,((20/10)<<4|(1%10))); //根据写日寄存器命令字，写入日的初始值
    WriteSet1302(0x88,((12/10)<<4|(2%10))); //根据写月寄存器命令字，写入月的初始值
    WriteSet1302(0x8c,((10/10)<<4|(4%10))); //根据写小时寄存器命令字，写入小时
的初始值
}
/*****
*****/

```

以下是对液晶模块的操作程序

```

*****
*****/
sbit RS=P2^0;           //寄存器选择位, 将RS 位定义为 P2.0 引脚
sbit RW=P2^1;           //读写选择位, 将RW 位定义为 P2.1 引脚
sbit E=P2^2;            //使能信号位, 将E 位定义为 P2.2 引脚
sbit BF=P0^7;           //忙碌标志位, 将BF 位定义为 P0.7 引脚
/*****
函数功能: 延时 1ms
(3j+2)*i=(3×33+2)×10=1010(微秒), 可以认为是 1 毫秒
*****/
void delay1ms()
{
    unsigned char i,j;
    for(i=0;i<10;i++)
        for(j=0;j<33;j++)
            ;
}
/*****
函数功能: 延时若干毫秒
入口参数: n
*****/
void delaynms(unsigned char n)
{
    unsigned char i;
    for(i=0;i<n;i++)
        delay1ms();
}
/*****
函数功能: 判断液晶模块的忙碌状态
返回值: result。result=1, 忙碌;result=0, 不忙
*****/
bit BusyTest(void)
{
    bit result;
    RS=0;           //根据规定, RS 为低电平, RW 为高电平时, 可以读状态
    RW=1;
    E=1;           //E=1, 才允许读写
    _nop_();        //空操作
    _nop_();
    _nop_();
    _nop_();        //空操作四个机器周期, 给硬件反应时间
    result=BF;      //将忙碌标志电平赋给 result
    E=0;           //将 E 恢复低电平
    return result;
}
/*****
函数功能: 将模式设置指令或显示地址写入液晶模块
入口参数: dictate
*****/
void WriteInstruction (unsigned char dictate)

```

```

{
    while(BusyTest()==1);    //如果忙就等待
    RS=0;                    //根据规定, RS 和 R/W 同时为低电平时, 可以写入指令
    RW=0;
    E=0;                     //E 置低电平写指令时, E 为高脉冲
                             // 就是让 E 从 0 到 1 发生正跳变, 所以应先置 "0"

    _nop_();
    _nop_();                 //空操作两个机器周期, 给硬件反应时间
    P0=dictate;              //将数据送入 P0 口, 即写入指令或地址
    _nop_();
    _nop_();
    _nop_();
    _nop_();                 //空操作四个机器周期, 给硬件反应时间
    E=1;                     //E 置高电平
    _nop_();
    _nop_();
    _nop_();
    _nop_();                 //空操作四个机器周期, 给硬件反应时间
    E=0;                     //当 E 由高电平跳变成低电平时, 液晶模块开始执行命令
}
/*****
函数功能: 指定字符显示的实际地址
入口参数: x
*****/
void WriteAddress(unsigned char x)
{
    WriteInstruction(x|0x80); //显示位置的确定方法规定为"80H+地址码 x"
}
/*****
函数功能: 将数据(字符的标准 ASCII 码)写入液晶模块
入口参数: y(为字符常量)
*****/
void WriteData(unsigned char y)
{
    while(BusyTest()==1);
    RS=1;                    //RS 为高电平, RW 为低电平时, 可以写入数据
    RW=0;
    E=0;                     //E 置低电平(根据表 8-6, 写指令时, E 为高脉冲,
                             // 就是让 E 从 0 到 1 发生正跳变, 所以应先置"0"
    P0=y;                    //将数据送入 P0 口, 即将数据写入液晶模块
    _nop_();
    _nop_();
    _nop_();
    _nop_();                 //空操作四个机器周期, 给硬件反应时间
    _nop_();
    _nop_();
    E=1;                     //E 置高电平
    _nop_();
    _nop_();
    _nop_();

```



```

        nop ();           //空操作四个机器周期, 给硬件反应时间
        nop ();
        E=0;             //当 E 由高电平跳变成低电平时, 液晶模块开始执行命令
    }
/*****
函数功能: 对 LCD 的显示模式进行初始化设置
*****/
void LcdInitiate(void)
{
    delaynms(15);         //延时 15ms, 首次写指令时应给 LCD 一段较长的反应时间
    WriteInstruction(0x38); //显示模式设置: 16×2 显示, 5×7 点阵, 8 位数据接口
    delaynms(5);          //延时 5ms , 给硬件一点反应时间
    WriteInstruction(0x38);
    delaynms(5);          //延时 5ms , 给硬件一点反应时间
    WriteInstruction(0x38); //连续三次, 确保初始化成功
    delaynms(5);          //延时 5ms , 给硬件一点反应时间
    WriteInstruction(0x0c); //显示模式设置: 显示开, 无光标, 光标不闪烁
    delaynms(5);          //延时 5ms , 给硬件一点反应时间
    WriteInstruction(0x06); //显示模式设置: 光标右移, 字符不移
    delaynms(5);          //延时 5ms , 给硬件一点反应时间
    WriteInstruction(0x01); //清屏幕指令, 将以前的显示内容清除
    delaynms(5);          //延时 5ms , 给硬件一点反应时间

}
/*****
以下是 1302 数据的显示程序
*****/
/*****
函数功能: 显示秒
入口参数: x
*****/
void DisplaySecond(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位
    j=x%10; //取个位
    WriteAddress(0x46);     //写显示地址, 将在第 2 行第 1 列开始显示
    WriteData(digit[i]);    //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);    //将十位数字的字符常量写入 LCD
    delaynms(50);          //延时 1ms 给硬件一点反应时间
}

/*****
函数功能: 显示分钟
入口参数: x
*****/
void DisplayMinute(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位

```

```

    j=x%10;//取个位
    WriteAddress(0x43);    //写显示地址,将在第2行第7列开始显示
    WriteData(digit[i]);   //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);   //将十位数字的字符常量写入 LCD
    delaynms(50);         //延时 1ms 给硬件一点反应时间
}
/*****
函数功能: 显示小时
入口参数: x
*****/
void DisplayHour(unsigned char x)
{
    unsigned char i,j;     //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x40);    //写显示地址,将在第2行第7列开始显示
    WriteData(digit[i]);   //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);   //将十位数字的字符常量写入 LCD
    delaynms(50);         //延时 1ms 给硬件一点反应时间
}
/*****
函数功能: 显示日
入口参数: x
*****/
void DisplayDay(unsigned char x)
{
    unsigned char i,j;     //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x0e);    //写显示地址,将在第2行第7列开始显示
    WriteData(digit[i]);   //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);   //将十位数字的字符常量写入 LCD
    delaynms(50);         //延时 1ms 给硬件一点反应时间
}
/*****
函数功能: 显示月
入口参数: x
*****/
void DisplayMonth(unsigned char x)
{
    unsigned char i,j;     //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x0b);    //写显示地址,将在第2行第7列开始显示
    WriteData(digit[i]);   //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);   //将十位数字的字符常量写入 LCD
    delaynms(50);         //延时 1ms 给硬件一点反应时间
}
/*****
函数功能: 显示年

```

入口参数: x

```

*****/
void DisplayYear(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位
    j=x%10; //取个位
    WriteAddress(0x08);     //写显示地址,将在第2行第7列开始显示
    WriteData(digit[i]);    //将百位数字的字符常量写入 LCD
    WriteData(digit[j]);    //将十位数字的字符常量写入 LCD
    delaynms(50);          //延时 1ms 给硬件一点反应时间
}

```

/******

函数功能: 主函数

*****/

```

void main(void)
{
    unsigned char second,minute,hour,day,month,year;      //分别储存苗、分、小
    时,日,月,年
    unsigned char ReadValue; //储存从 1302 读取的数据
    LcdInitiate();           //将液晶初始化
    WriteAddress(0x01);     //写 Date 的显示地址,将在第1行第2列开始显示
    WriteData('D');         //将字符常量写入 LCD
    WriteData('a');         //将字符常量写入 LCD
    WriteData('t');         //将字符常量写入 LCD
    WriteData('e');         //将字符常量写入 LCD
    WriteData(':');          //将字符常量写入 LCD

    WriteAddress(0x06);     //写年月分隔符的显示地址,显示在第1行第9列
    WriteData('2');         //将字符常量写入 LCD
    WriteAddress(0x07);     //写年月分隔符的显示地址,显示在第1行第9列
    WriteData('0');         //将字符常量写入 LCD

    WriteAddress(0x0a);     //写年月分隔符的显示地址,显示在第1行第9列
    WriteData('-');         //将字符常量写入 LCD
    WriteAddress(0x0d);     //写月日分隔符的显示地址,显示在第1行第12列
    WriteData('-');         //将字符常量写入 LCD
    WriteAddress(0x42);     //写小时与分钟分隔符的显示地址,显示在第2行第6列
    WriteData(':');         //将字符常量写入 LCD
    WriteAddress(0x45);     //写分钟与秒分隔符的显示地址,显示在第2行第9列
    WriteData(':');         //将字符常量写入 LCD
    Init_DS1302();          //将 1302 初始化
    while(1)
    {
        ReadValue = ReadSet1302(0x81); //从秒寄存器读数据
        second=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
        DisplaySecond(second);         //显示秒
        ReadValue = ReadSet1302(0x83); //从分寄存器读
        minute=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    }
}

```

```

DisplayMinute(minute);          //显示分
ReadValue = ReadSet1302(0x85);  //从分寄存器读
hour=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
DisplayHour(hour);              //显示小时
ReadValue = ReadSet1302(0x87);  //从分寄存器读
day=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
DisplayDay(day);                //显示日
ReadValue = ReadSet1302(0x89);  //从分寄存器读
month=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
DisplayMonth(month);            //显示月
ReadValue = ReadSet1302(0x8d);  //从分寄存器读
year=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
DisplayYear(year);              //显示年
}
}

```

实例二 基于 ADC0832 和 LCD1602 的数字电压表设计

1. 实例说明

要求用单片机和 ADC0832 设计一个 0~5V 的直流电压表，测量由 CH0 输入的直流电压，并由 LCD1602 显示测量的直流电压值。关于 ADC0832 的使用请参考其手册。

2. 仿真电路

基于 ADC0832 和 LCD1602 的数字电压表设计仿真电路如图 7.2 所示。

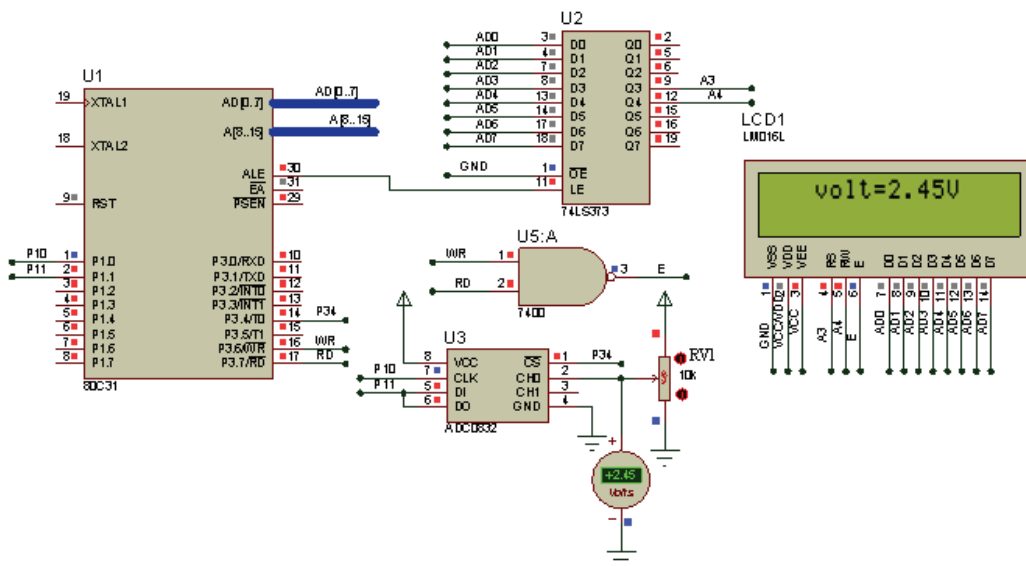


图 7.2 基于 ADC0832 和 LCD1602 的数字电压表设计仿真电路

3. 程序设计

```
#include<reg51.h>
#include<absacc.h>
//#include<lcd.h>
#define CMD_WR XBYTE[0x00]
#define DATA_WR XBYTE[0x08]
#define BUSY_RD XBYTE[0x10]
#define DATA_RD XBYTE[0x18]
#define CLS 0x01
#define HOME 0x02
#define SETMODE 0x06
#define SETVISIBLE 0x08
#define SHIFT 0x10
#define SETFUNCTION 0x38
#define SETCGADDR 0x40
#define SETDDADDR 0x80
char lcd[]={"volt=1.34V6789ABCDEF"};
void busy(void);
void Delay(unsigned char j);
void lcdini(void)
{
    Delay(20);
    busy();
    CMD_WR=SETFUNCTION;
    busy();
    CMD_WR=SETVISIBLE;
    busy();
    CMD_WR=CLS;
    busy();
    CMD_WR=SETMODE;
    busy();
    CMD_WR=0x0c;
}

void busy(void)
{
    Delay(5);
}

void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //耗时 1 毫秒
    }
}

sbit CS=P3^4; //将 CS 位定义为 P3.4 引脚
```

```

sbit CLK=P1^0;    //将 CLK 位定义为 P1.0 引脚
sbit DIO=P1^1;    //将 DIO 位定义为 P1.1 引脚

void delay1(unsigned char j)
{
    while(j--);
}
/*****
函数功能：将模拟信号转换成数字信号
*****/
unsigned char A_D()
{
    unsigned char i,dat=0;
    CS=1;    //一个转换周期开始
    CLK=0;    //为第一个脉冲作准备
    CS=0;    //CS 置 0，片选有效
    DIO=1;    //DIO 置 1，规定的起始信号

    CLK=1;    //第一个脉冲
    CLK=0;    //第一个脉冲的下降沿，此前 DIO 必须是高电平
    DIO=1;    //DIO 置 1，通道选择信号

    CLK=1;    //第二个脉冲，第 2、3 个脉冲下沉之前，DI 必须跟别输入两位数据用于选择通道，
    这里选通道 CH0
    CLK=0;    //第二个脉冲下降沿
    DIO=0;    //DI 置 0，选择通道 0

    CLK=1;    //第三个脉冲
    CLK=0;    //第三个脉冲下降沿
    DIO=1;    //第三个脉冲下沉之后，输入端 DIO 失去作用，应置 1

    CLK=1;    //第四个脉冲
    for(i=0;i<8;i++)    //高位在前
    {
        CLK=1;    //第四个脉冲
        CLK=0;
        dat<<=1;    //将下面储存的低位数据向右移
        dat|=(unsigned char)DIO;    //将输出数据 DIO 通过或运算储存在 dat 最低位
    }
    CS=1;    //片选无效
    return dat;    //将读出的数据返回
}
/*****
函数功能：主函数
*****/
void main(void)
{
    unsigned char k,a,zh,xiao;
    lcdini();
    while(1)
    {

```

```
a=A D();  
zh=a/51;  
lcd[5]=zh+0x30;  
xiao=(a%51)*100/51;  
lcd[7]=(xiao/10)+0x30;  
lcd[8]=(xiao%10)+0x30;  
busy();  
CMD_WR=0x83;  
for(k=0;k<10;k++)  
{  
    busy();  
    DATA_WR=lcd[k];  
    delay1(200);  
}  
}
```

本章小结

本章主要就单片机应用系统的一般开发过程作了系统的介绍。其中包括前期硬件和软件框架的构建、相关元器件的选型、相关开发环境的使用，以及相关开发工具的选择和使用。最后简要论述了系统的优化设计，包括系统可靠性设计、抗干扰设计，以及系统的自诊断等。

读者通过本章的学习，对单片机应用系统设计的基本步骤和方法有了一个初步认识，知道在单片机的应用开发过程中要注意哪些细节，为后续实际的单片机应用系统开发打下基础。


习题七

一、简答题

1. 简述单片机应用系统的设计步骤。
2. 简述单片机应用系统的开发与调试过程。
3. 如何抑制干扰源。

二、设计编程题

1. 设计一个温度控制系统，设计上限和下限温度，超过限定温度时报警，实现对温度的实时测量与显示。

2. 用 AT89C52 设计一个智能稳压电源。 

第 8 章 常用开发仿真软件 Keil c 和 Proteus 简介

学习目标

掌握单片机开发工具 Keil C 和 Proteus 软件的使用，能在通用计算机上进行单片机软件和硬件的仿真，模拟单片机的程序运行过程，在单片机程序的运行过程中随时观测单片机的运行状态。

重点难点

Keil c 和 Proteus 软件的使用。

8.1 知识结构

8.1.1 Keil C 编译器使用简介

下面以不断从串口输出“Huanghuai University”字符串为例说明 Keil C 的使用。程序清单：

```
#include <reg51.h>
#include <stdio.h>
void main(void)
{
    SCON = 0x50; // 串口方式 1，允许接收
    TMOD = 0x20; // 定时器 1 定时方式 2
    TCON = 0x40; // 设定定时器 1 开始计数
    TH1 = 0xE8; // 11.0592MHz 1200 波特率
    TL1 = 0xE8;
    TI = 1;
    TR1 = 1; // 启动定时器
    while(1)
    {
        printf ("Huanghuai University \n"); // 显示 Huanghuai University
    }
}
```

(1) 单击【Project】菜单，选择弹出的下拉式菜单中的【New Project】，如图 8.1 所示。接着弹出一个标准 Windows 文件对话框，如图 8.2 所示。在“文件名”中输入第一个 C 程序项目名称，这里用“sample”，保存后的文件扩展名为 uv2，这是 KEIL uVision2

项目文件扩展名，以后可以直接点击此文件以打开先前做的项目。

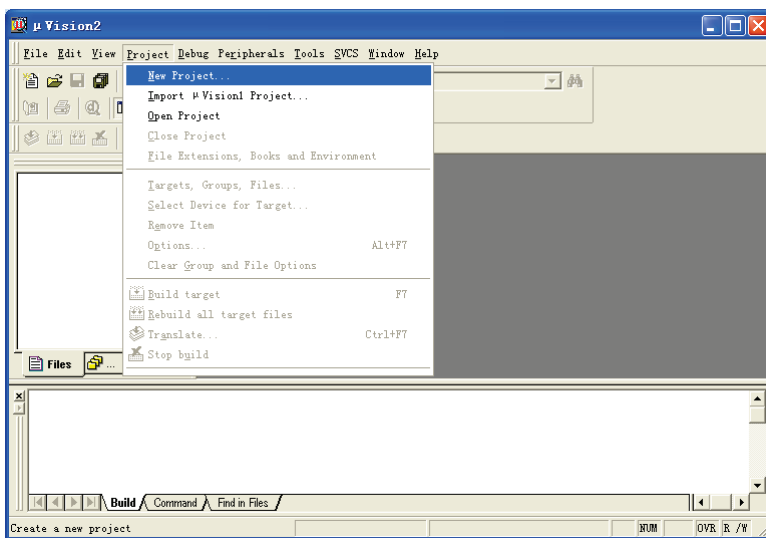


图 8.1 New Project 菜单

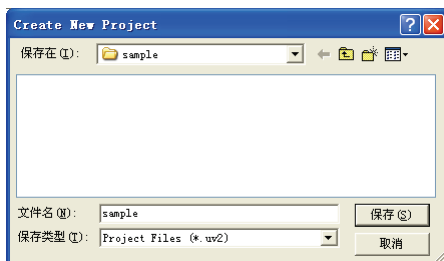


图 8.2 Create New Project 对话框

(2) 选择所需要的单片机，这里选择常用的 Ateml 公司的 AT89C51。此时屏幕如图 8.3 所示。单击【确定】按钮后，就可以进行程序的编写了。

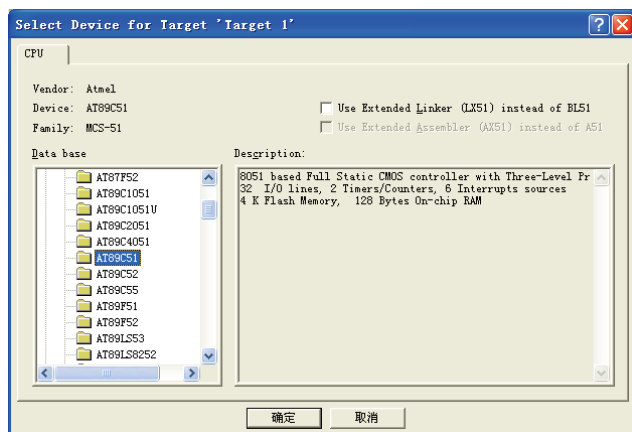



图 8.3 选取芯片

(3) 在项目中创建新的程序文件或加入旧程序文件。如果没有现成的程序,就要新建一个程序文件。在 KEIL 中有一些程序的例子,这里我们就以本节开始的一个 C 程序为例介绍如何新建一个 C 程序并加到该项目中。单击图 8.4 中的新建文件(New file)的快捷按钮,出现一个新的文字编辑窗口 Text1。这个操作也可以通过菜单 File→New 或快捷键【Ctrl+N】来实现。此时光标已出现在文本编辑窗口,就可以编写程序了。

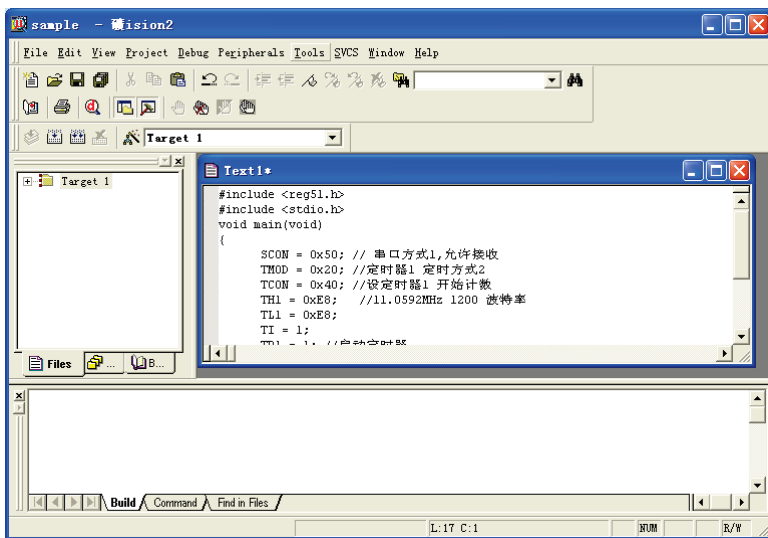



图 8.4 新建程序文件

(4) 单击图 8.4 中的按钮保存新建的程序,也可以用菜单 File→Save 或快捷键【Ctrl+S】进行保存。因为是新文件,所以保存时会弹出像图 8.5 所示的文件操作窗口,把第一个程序命名为 test1.c,保存在项目所在的目录中,这时会发现程序单词有了不同的颜色,说明 KEIL 的 C 语法检查生效了。单击“Target 1”前面的【+】,如图 8.6 所示,鼠标在屏幕左边的“Source Group1”文件夹图标上右击弹出菜单,在这里可以做在项目中增加、减少文件等操作。选择“Add Files to Group ‘Source Group 1’”文件,弹出文件窗口,如图 8.7 所示,选择刚刚保存的文件“text1.c”,单击【ADD】按钮,再单击【Close】。程序文件已加到项目中了。这时在 Source Group1 文件夹图标左边出现了一个小“+”号说明,文件组中有了文件,点击它可以展开查看。

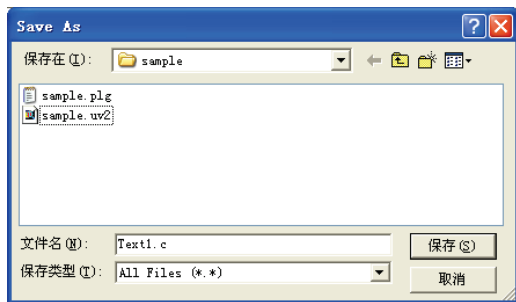


图 8.5 “另存为”对话框

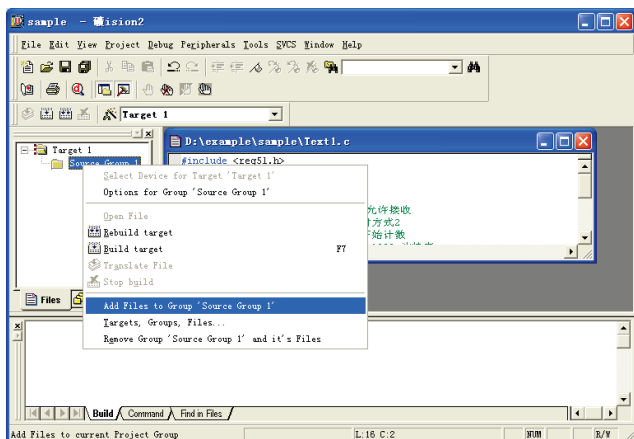


图 8.6 项目中添加源文件窗口

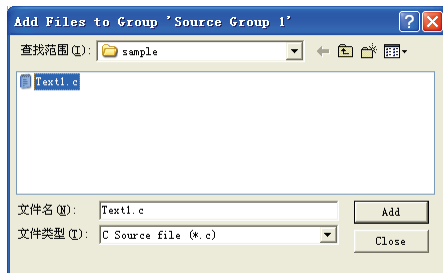







图 8.7 “Add Files to Group ‘Source Group 1’”对话框

(5) 编译：如图 8.8 所示，图中工具栏上有三个编译按钮，不同的是用于编译单个文件。是编译当前项目，如果先前编译过一次之后文件没有编辑改动，这时再点击是不会再次重新编译的。是重新编译，每点击一次均会再次编译链接一次，不管程序是否有改动。在右边的是停止编译按钮，只有点击了前三个中的任一个，“停止”按钮才会生效。也可以使用【Project】菜单中的【Build target】、【Rebuild all target files】、【Translate】，在窗口下方可以看到编译的错误信息和使用的系统资源情况等，以后查错就靠它了。是一个放大镜的按钮，这就是开启\关闭调试模式的按钮，在菜单【Debug】中为【Start/Stop Debug Session】，快捷键为【Ctrl+F5】。

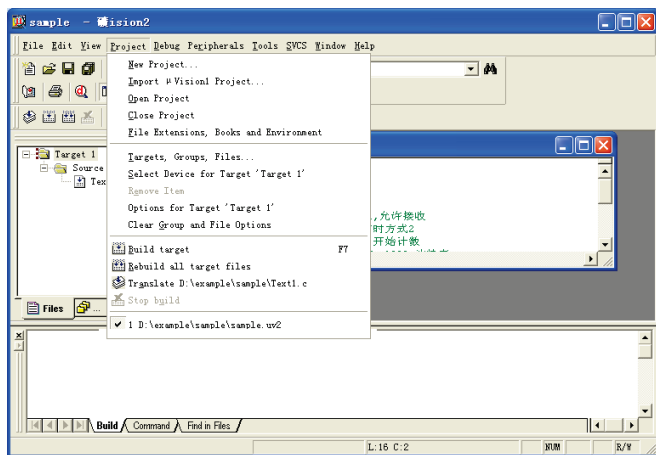







图 8.8 编译程序

(6) 调试：图 8.9 中为运行，当程序处于停止状态时才有效，为停止，程序处于运行状态时才有效。是复位，模拟芯片的复位，程序回到最开头处执行。按可以打开串行调试窗口（Serial #1），这个窗口可以看到从 51 芯片的串行口输入输出的字符，本项目也正是在这里看运行结果。这些命令在【Debug】菜单中也有，这里不再一一介绍，大家不妨找找看。首先按打开串行调试窗口，再按运行键，这时就可以看到串行调试窗口

中不断地打印“Huanghuai University”。这样就完成了第一个 C 项目。最后要停止程序运行回到文件编辑模式中,就要先按停止按钮再按开启\关闭调试模式按钮。然后就可以进行关闭 KEIL 等相关操作了。

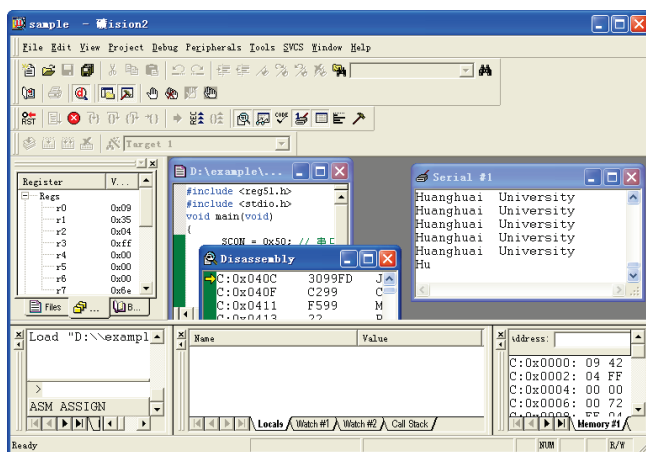


图 8.9 调试运行程序

(7) 生成 HEX 文件: 找到前面建的 sample.Uv2 文件, 打开先前的项目。然后右击图 8.10 中的“Target 1”项目文件夹, 弹出项目功能菜单, 选“Options for Target ‘Target1’”, 弹出项目选项设置窗口, 同样先选中项目文件夹图标, 这时在【Project】菜单中也有一样的菜单可选。打开项目选项窗口, 转到【Output】选项页, 如图 8.11 所示, 图中单击【Select Folder for Objects..】选择编译输出的路径。按图指示设置编译输出生成的文件名, 选择复选按钮决定是否要创建 HEX 文件, 选中它就可以输出 HEX 文件到指定的路径中。单击【确定】按钮后, 再将重新编译项目一次, 很快在编译信息窗口中就显示 HEX 文件创建到指定的路径中了, 如图 8.12 所示。这样就可使用编程器所附带的软件去读取芯片了, 再用实验板看结果, 也可用于 Proteus 仿真, 由于编程器或仿真器品种繁多, 具体方法看说明书, 这里不做讨论。

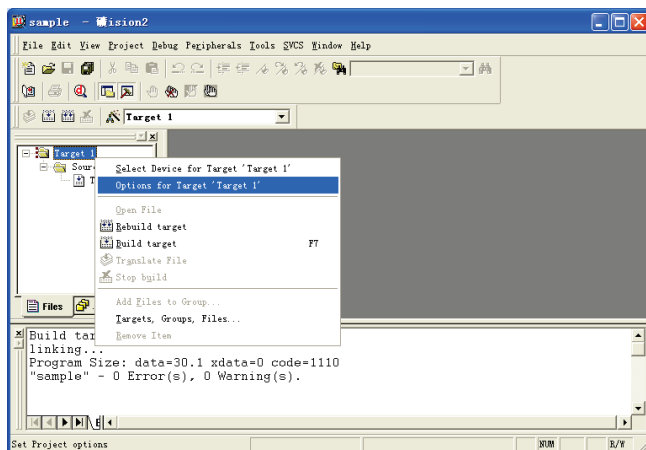


图 8.10 项目功能菜单

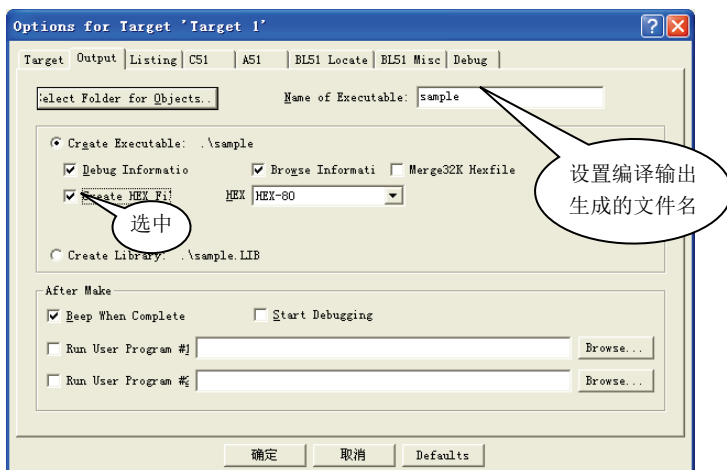


图 8.11 项目选项窗口

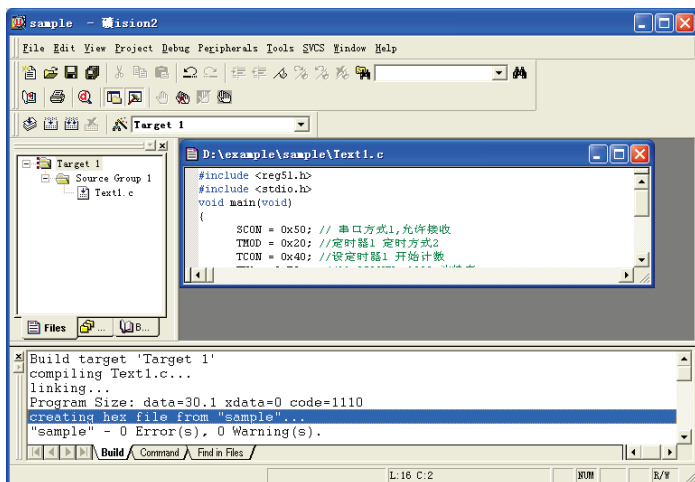


图 8.12 编译信息窗口

8.1.2 Proteus 仿真软件使用简介

1. 界面介绍

双击桌面上的 ISIS 7 Professional 图标或者单击屏幕左下方的“开始”→“程序”→“Proteus 7 Professional”→“ISIS 7 Professional”，出现如图 8.13 所示屏幕，表明进入 Proteus ISIS 集成环境。

进入之后的界面类似如图 8.14 所示。图中已经标注各个部分的作用。



图 8.13 Proteus ISIS 集成环境

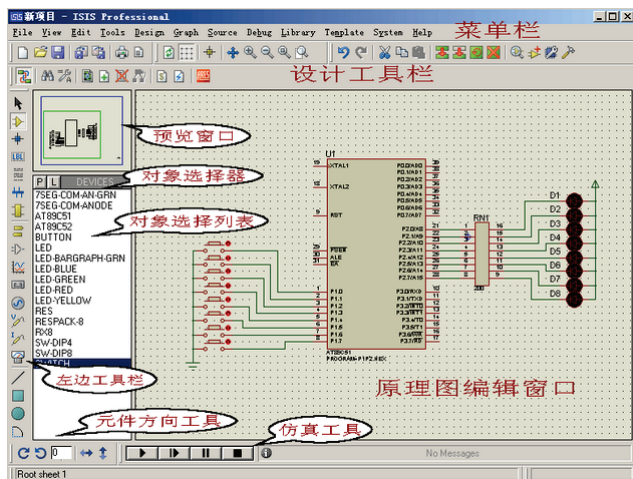


图 8.14 ISIS 主窗口

2. 一个小项目的设计过程

(1) 建立新项目

启动软件之后，首先，新建一个项目。

点击菜单：File→New Design，如图 8.15 所示，即可出现如图 8.16 所示的对话框，以选择设计模板。一般选择 A4 图纸即可，点击 OK，关闭对话框，完成设计图纸的模板选择，出现一个空白的设计空间。

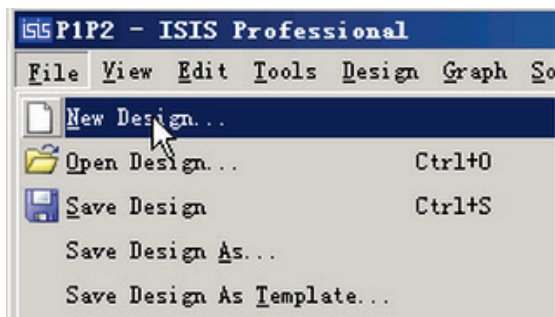


图 8.15 新设计

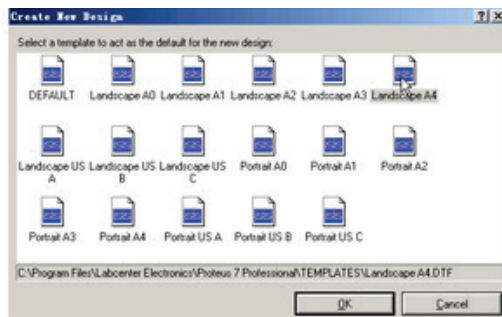


图 8.16 选模板

这时设计名称为 UNTITLED (未命名)，可以点击菜单 file→save design 来给设计命名，也可以在设计的过程中任何时候命名。

(2) 调入元件

在新设计窗口中，点击对象选择器上方的按钮 P (如图 8.17 所示)，即可进入元件拾取对话框，如图 8.18 所示。

在图 8.18 所示的对话框左上角，有一个 Keywords 输入框，可以在此输入要用的元件名称 (或名称的一部分)，右边出现符合输入名称的元件列表。这里要用的单片机是 AT89C51，输入 AT89C，就出现一些元件，选中 AT89C51，双击，就可以将它调入设计窗口的元件选择器。

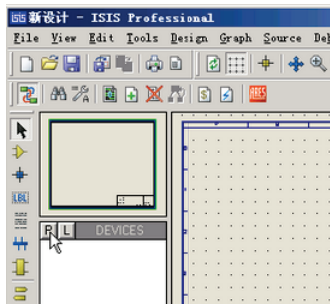


图 8.17 调入元件

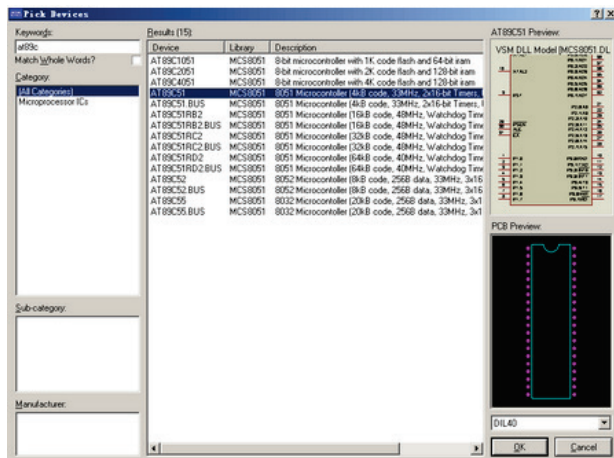


图 8.18 查找元件

在 Keywords 中重新输入要用到的元件, 比如 LED, 双击需要用的具体元件, 比如 LED-YELLOW, 调入。继续输入, 调入, 直到够用。单击 OK 按钮, 关闭对话框。以后如果需要其他元件, 还可以再次调入。元件调入之后的情形类似图 8.14 中的对象选择列表所示。

这次要用到的元件列表如下:

AST89C51	单片机
LED-YELLOW	发光二极管-黄色
RX8	8 电阻排 200 欧姆
BUTTON	按钮

以上元件就够用了, 其他多余的只是供选用。比如发光二极管可以选用其他颜色, 按钮也可以使用 SWITCH 代替或者使用 DIP-SW8 代替, 电阻排也可以使用单个电阻器 RES 来代替。

(3) 设计原理图

① 放置元件。

在对象选择器中的元件列表中, 单击所用元件, 再在设计窗口单击, 出现所用元件的轮廓, 并随鼠标移动, 找到合适位置, 单击, 元件被放到当前位置。至此, 一个元件放置好了, 继续放置要用到的其他元件。

② 移动元件。

如果要移动元件的位置, 可以先右击元件, 元件颜色变红, 表示被选中, 然后拖动到需要的位置放下即可。放下后仍然是红色, 还可以继续拖动, 直到位置合适, 在空白处单击, 取消选中。

③ 移动多个元件。

如果几个元件要一起移动, 可以先把它们都选中, 然后移动。选中多个元件的方法是, 在空白处开始, 单击并拖动, 出现一个矩形框, 让矩形框包含需要选中的元件再放开, 就可以了 (参看图 8.19 所示)。如果选择不合适, 可以在空白处单击, 取消选中, 然后重新选择。移动元件的目的主要是为了便于连线, 当然也要考虑美观。

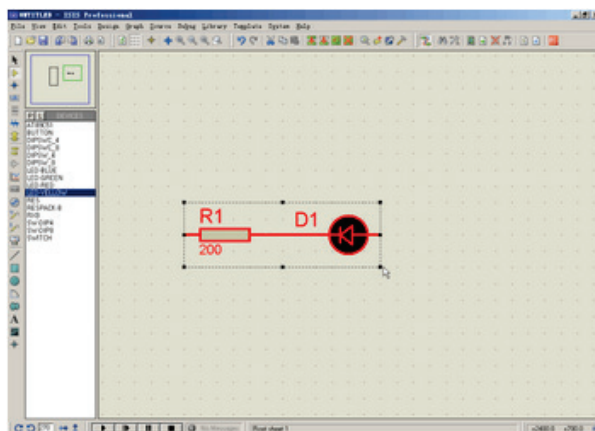
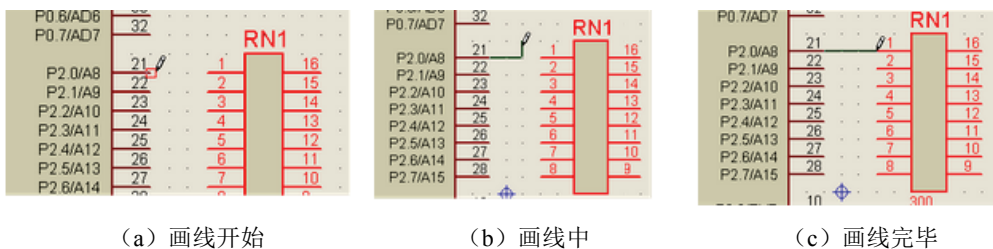


图 8.19 选中多个元件

④ 连线。

就是把元件的引脚按照需要用导线连接起来。方法是在开始连线的元件引脚处单击（光标接近引脚端点附近会出现红色小方框，这时就可以了），移动光标到另一个元件引脚的端点，单击即可。移动过程中会有一根线跟随光标延长，直到单击才停住，如图 8.20 所示。



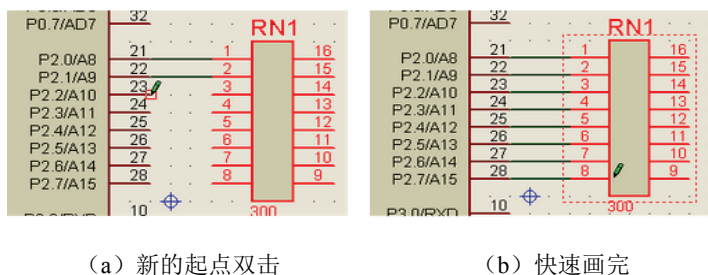
(a) 画线开始

(b) 画线中

(c) 画线完毕

图 8.20 画线过程

在第一根线画完后，第二根线可以自动复制前一根线，在一个新的起点双击即可，如图 8.21 所示。



(a) 新的起点双击

(b) 快速画完

图 8.21 自动复制前一根线

注意

如果第二根线形状与第一根不同，那可不能自动复制，否则会很麻烦。

⑤ 修改元件参数。

电阻器电容器等元件的参数可以根据需要修改。比如限流电阻器的电阻值应该在 200 欧姆到 500 欧姆之间，上拉电阻器应该在几千欧姆。

以修改限流电阻排为例，先单击或右击该元件以选中，然后再单击，出现对话框如图 8.22 所示。在 Component Value: 后面的输入框中输入电阻值 200（单位欧姆），然后单击 OK 按钮确认并关闭对话框，电阻值设置完毕。

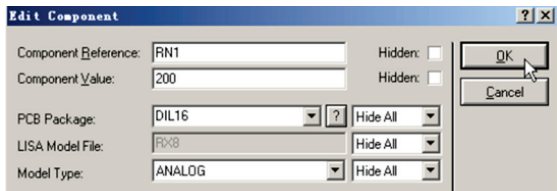

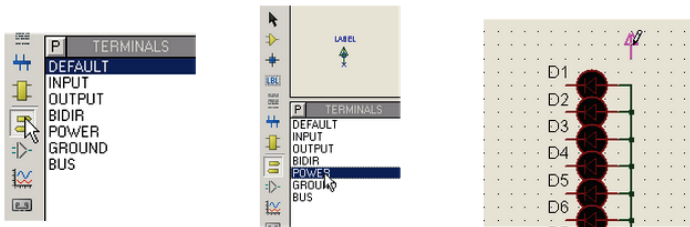


图 8.22 修改电阻值

⑥ 添加电源和接地符号。

在左边工具栏单击终端图标，即可出现可用的终端，如图 8.23 (a) 所示。

在对象选择器中的对象列表中，单击 POWER，如图 8.23 (b) 所示，在预览窗口出现电源符号，在需要放置电源的地方单击，即可放置电源符号，如图 8.23 (c) 所示。放置之后，就可以连线了。



(a) 选择端口


(b) 选择电源符号

(c) 放置电源符号

图 8.23 添加电源和地线

放置接地符号（地线）的方法与放置电源类似，在对象选择列表中单击 GROUND，然后在需要接地符号的地方单击，就可以了。

注意

放置电源接地之后，如果又需要放置元件，应该先单击左边工具栏元件图标，就会在对象列表中出现我们从元件库中调出来的元件。

按照图 8.14 所示的原理图，还需要放置按键，放置接地符号，连线，最终完成总的原理图设计。

8.2 学习实例

实例一 通过 P1.0 输出周期为 20ms 的方波信号

1. 实例说明

要求让 P1.0 输出周期为 20ms 的方波信号并用虚拟示波器观察波形。要注意虚拟示波器的使用。

2. 仿真电路

方波信号仿真电路如图 8.24 所示。

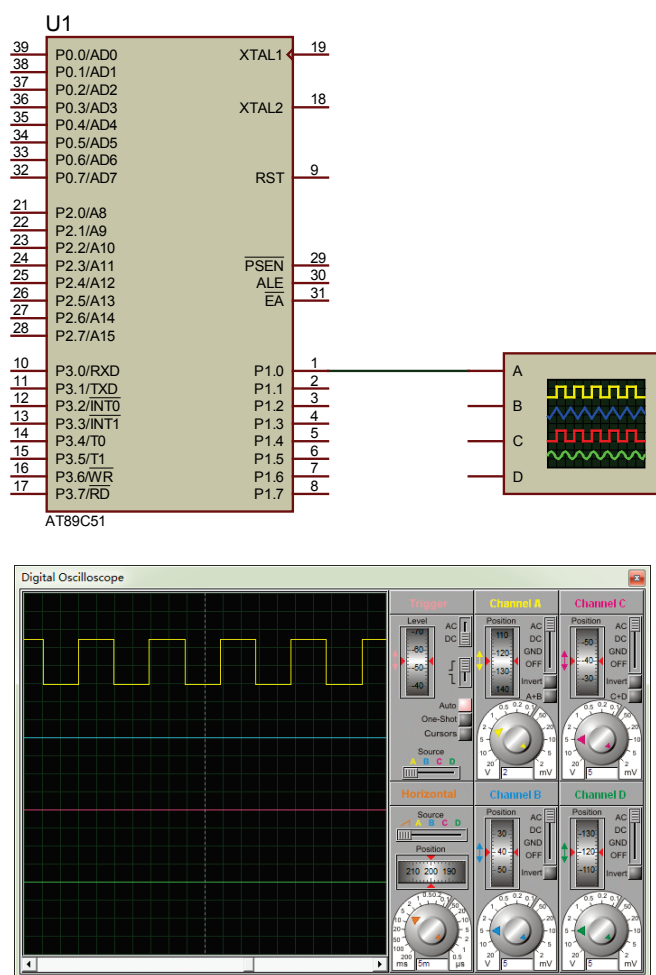


图 8.24 通过 P1.0 输出周期为 20ms 的方波信号仿真电路

3. 程序设计

```
#include<reg51.h>
void delay10ms(int n) //延时 10ms
{
    int i=0,j;
    while(n--)
    {
        for (i=0;i<10;i++)
        {
            for(j=0;j<120;j++);
        }
    }
}
```

```

    }
void main()
{
    while(1)
    {
        delay10ms(1);
        P1=0xfe; //P1 口输出为 0
        delay10ms(1);
        P1=0xff; //P1 口输出为 1
    }
}

```

实例二 计单个按键次数并显示

1. 实例说明

要求在 P1.0 接一按钮，编程对其按键次数进行计数，并把计数值通过 P2 和 P3 口的三位 BCD 码 7 段显示器显示出来。这里用的 7 段显示器，只要在输入端输入要显示数字的 BCD 码即可。

2. 仿真电路

计单个按键次数并显示仿真电路如图 8.25 所示。

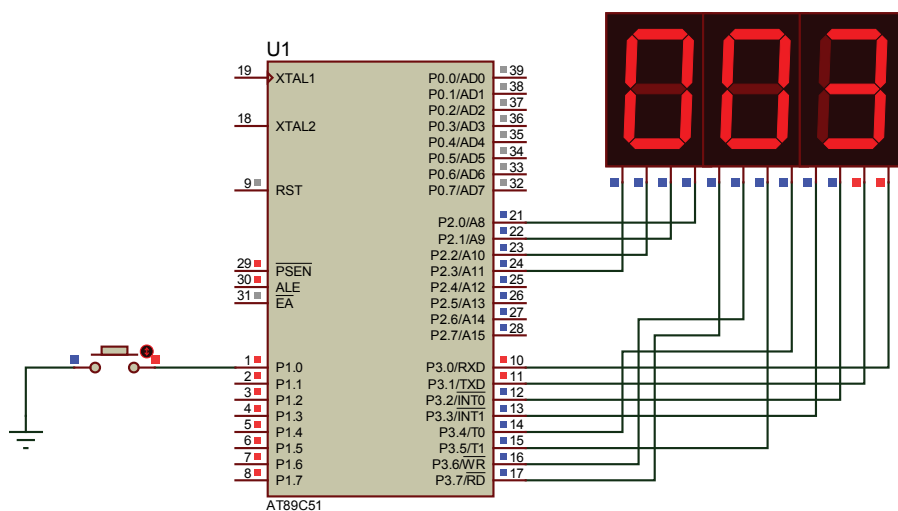


图 8.25 计单个按键次数并显示仿真电路

3. 程序设计

汇编程序如下：

```

ORG 0000H ; 起始地址为 0
    LJMP MAIN ; 转主程序
ORG 0030H
MAIN:MOV A, #0FFH;

```

```

    MOV R0,#40H
    LCALL BINBCD
    MOV P3,@R0
    DEC R0
    MOV P2,@R0
    SJMP $
BINBCD:MOV B,#64H
        DIV AB
    MOV @R0,A
    INC R0
    MOV A,#0AH
    XCH A,B
    DIV AB
    SWAP A
    ADD A,B
    MOV @R0,A
    RET

```

C 程序如下:

```

#include<reg51.h>
#include<absacc.h>
sbit S1=P1^0;
void main(void)
{
    unsigned char a,bai,shi,ge;
    a=0;
    P2=0;
    P3=0;
    while(1)
    {
        if(!S1)
        {
            while(!S1); //松手检测
            a=a+1;
            bai=a/100;
            shi=a%100/10;
            ge=a%10;
            P2=bai;
            P3=shi<<4|ge;
        }
    }
}

```

本章小结

本章通过实例主要介绍了用 Keil 集成开发环境进行项目程序的编辑、编译、链接和调试及用 Proteus 绘制电路图和仿真的方法、步骤。

习题三

一、填空题

1. Keil C51 软件中, 工程文件的扩展名是_____, 编译连接后生成可烧写的文件扩展名是_____。
2. Keil μ Vision3 集成开发环境中, 编译当前文件的快捷键为_____。
3. 在 Keil μ Vision3 编译系统中, 支持的 8051 系列单片机存储模式共有如下三种: _____、_____和_____。
4. Proteus VSM 主要特点是能把_____作用在处理器上, 并和该处理器的任何模拟和数字器件进行_____仿真。
5. 要把编好的 C51 程序生成可执行文件, 要经过_____和_____两个步骤。

二、选择题

1. 以下哪些不是 Keil μ Vision3 集成开发环境的特点 ()。
A. Windows 界面风格 B. 支持汇编语言和 C51 语言
C. 支持桌面程序的开发 D. 丰富的仿真调试功能
2. 用来设置输出 HEX 文件的命令标签页为下面哪项? ()
A. Debug 标签页 B. Output 标签页
C. Target 标签页 D. Device 标签页
3. 在 Keil 里开发 8051 程序的第一步是什么? ()
A. 调试与仿真 B. 产生执行文件
C. 组建程序 D. 打开或新建项目文件
4. 在 Keil 里要导入 C 源程序, 应如何操作? ()
A. 运行 “File” | “New” 命令 B. 双击 “Source Group 1”
C. 运行 “Project” | “New” 命令 D. 单击 “Source Group 1”

三、简答题

1. 如何实现用 Keil C51 与 Proteus 原理图进行高级语言源代码仿真调试?
2. Poteus 软件有哪些功能?
3. KEIL 软件在调试程序时提供了多个窗口, 主要包括哪几个?

四、设计题

1. 设计一个流水灯程序, 编译调试并最终生成 HEX 文件。
2. 用 Proteus 软件和 Keil 软件联合调试一个流水灯项目, 仿真出结果。
3. 按要求实现下列操作:
 - (1) 先搭建一个 “8051 基本 I/O 实验” 的仿真电路, 该单片机系统功能是一个开关闭合后, 有一个对应的 LED 指示灯亮。
 - (2) 利用已搭建的电路运行 “查表指令程序”, 实验要求输出的花样按照一个常数表的数值改变。

第9章 单片机实验指导

实验一 P1 口实验

一、实验要求

- (1) P1 口作为输出口，接 8 只发光二极管，编写程序，使发光二极管循环点亮。
- (2) P1 口作为输入口，接 8 个扭子开关，以实验台上 74LS273 做输出口，编写程序读取开关状态，将此状态，在发光二极管上显示出来。

二、实验目的

- (1) 学习 P1 口的使用方法。
- (2) 学习延时子程序的编写和使用。

三、实验说明

P1 口为准双向口，P1 的每一位都能独立地定义为输出线或输入线，作为输入的口线，必须向锁存器的相应位写入“1”，该位才能作为输入。8031 中所有口锁存器在复位时均置为“1”，如果后来在口锁存器写入过“0”，在需要时应写入一个“1”，使它再成为一个输入。

可以用第二个实验做一下试验。先按要求做好程序并调试成功后，可将 P1 口锁存器置“0”，此时将 P1 做输入口，会有什么结果。

再来看一下延时程序的实现。现常用的有两种方法，一是用定时器中断来实验，二是用指令循环来实现。在系统时间允许的情况下可以采用后一种方法。

本实验系统晶振为 6.144MHz，则一个机器周期约为 0.2 μ s。现要写一个延时 0.1s 的程序，可大致写出如下：

```
DEL1: MOV R2,#200
DEL2: MOV R3,#126
DEL3: DJNZ R3,DEL3
      DJNZ R2,DEL2
      RET
```

四、仿真实验电路

用 P1 口做为输出口仿真电路如图 9.1 所示，用 P1 口做为输入口，仿真电路如图 9.2 所示。

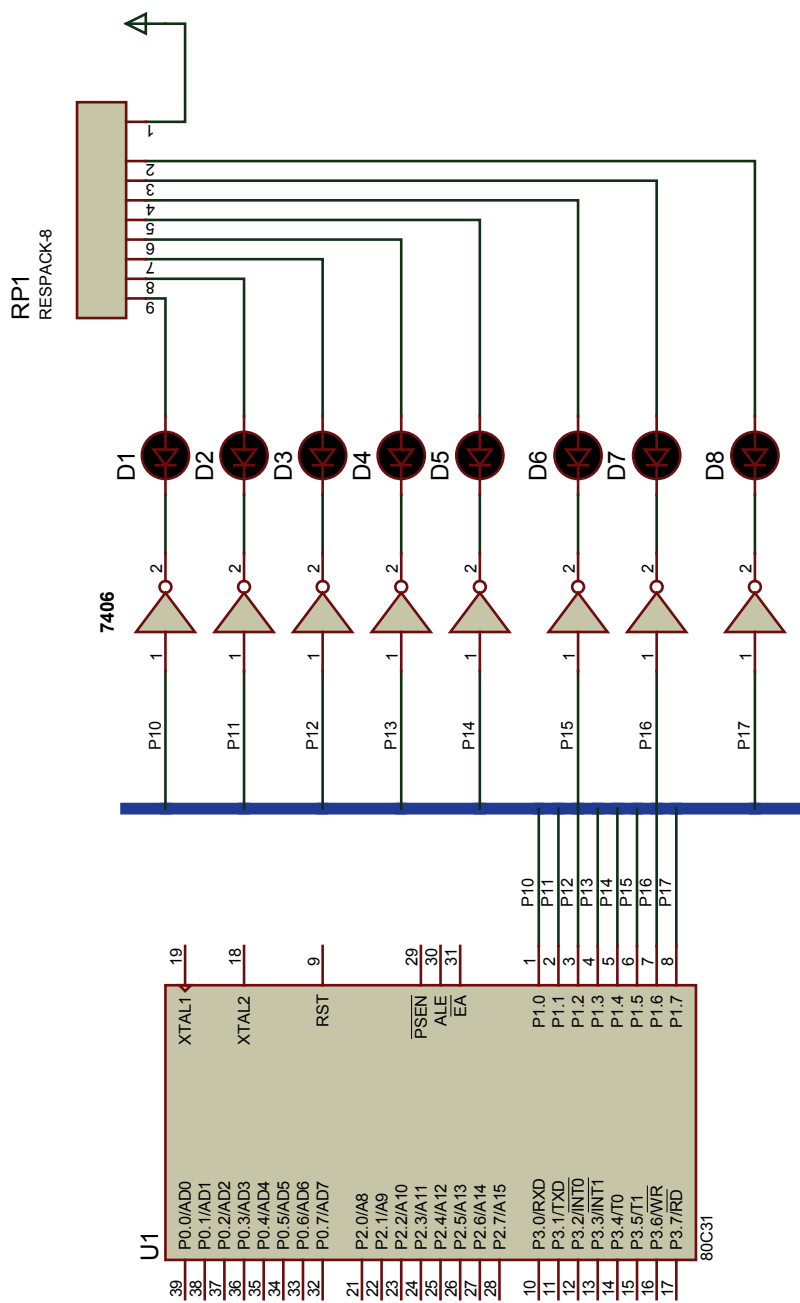


图 9.1 用 P1 口作为输出仿真电路

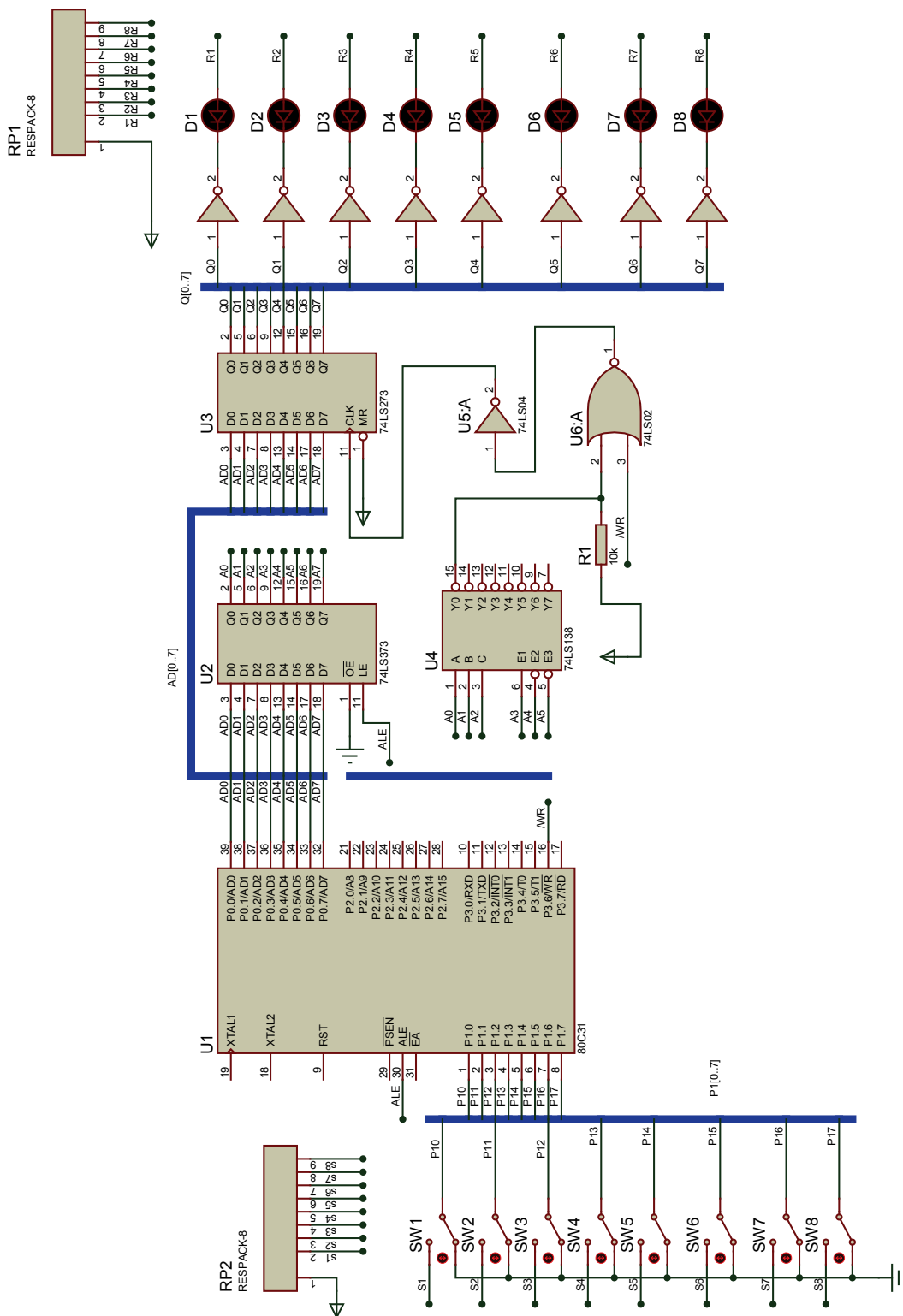


图 9.2 用 P1 口做为输入口仿真电路

五、参考程序清单

(一) 程序一

汇编程序: e9_1_1.asm

```
ORG 0000H
    LJMP    START
    ORG 200H
START:MOV A,#01H;置初值
LOOP: MOV P1,A;送 P1 口, 第 1 个 LED 灯亮
      MOV R1,#10;以下延时
DEL1:MOV R2,#200
DEL2:MOV R3,#126
DEL3:DJNZ R3,DEL3
      DJNZ R2,DEL2
      DJNZ R1,DEL1
      RL  A;左移
      LJMP LOOP;循环
END
```

C51 程序: e9_1_1.c

```
#include<reg51.h>
void delay10ms(int n)
{
    int i=0,j;
    while(n--)
    {
        for (i=0;i<10;i++)
            for(j=0;j<120;j++)
            {
            }
    }
}
void main()
{
    unsigned char i=0;
    P1=0x01;
    while(1)
    {
        delay10ms(100);
        P1=P1<<1;
        if(i==8)
        {
            i=0;
            P1=0x01;
        }
        i=i+1;
    }
}
```

（二）程序二

汇编语言：e9_1_2.asm

```

ORG 0000H
    SJMP MAIN
    ORG 0080H
MAIN: MOV P1,#0FFH
START: MOV A,P1
        MOV DPTR,#0FFC8H
        MOVX @DPTR,A
        LJMP START
        END
C51 程序: e9_1_2.c
#include<reg51.h>
#include<absacc.h>
#define P273 XBYTE[0xffc8]
main()
{
    P1=0xff;
    while(1)
    {
        char a;
        a=P1;
        P273=P1;
    }
}

```

实验二 交通灯控制实验

一、实验要求

用汇编和 C51 编写模拟交通灯的控制程序。

二、实验目的

- (1) 学习汇编和 C51 程序的编写方法。
- (2) 学习模拟交通灯控制的实现方法。

三、实验说明

要完成本实验，首先必须了解交通路灯的亮灭规律。假设交通灯的亮灭规律为：初始态是两个路口的红灯全亮，之后，东西路口的红灯亮，南北路口的绿灯亮，南北方向通车。延时一段时间后，南北路口绿灯灭，黄灯开始闪烁。闪烁若干次后，南北路口红灯亮，而同时东西路口的绿灯亮，东西方向开始通车。延时一段时间后，东西路口的绿灯灭，黄灯开始闪烁。闪烁若干次后，再切换到南北路口方向，重复上述过程。

四、仿真实验电路

交通灯控制实验仿真电路如图 9.3 所示。

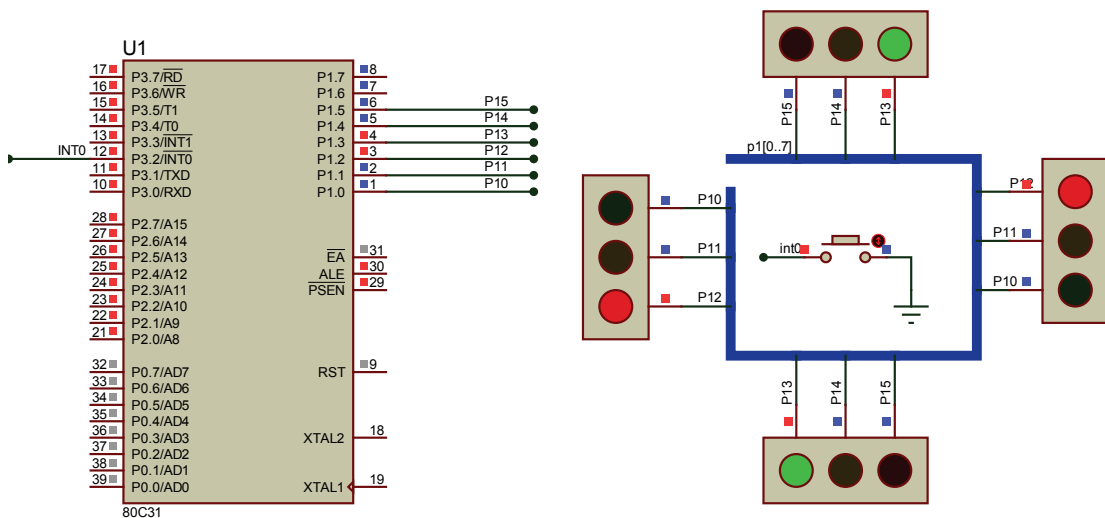


图 9.3 交通灯控制实验仿真电路

五、参考程序清单

汇编语言: e9-2.asm

```

    ORG 0000H
    LJMP MAIN
    ORG 0003H
    LJMP INT0P
    ORG 0100H
MAIN:SETB EA
    SETB EX0
    SETB IT0
LOOP:MOV P1,#24H
    LCALL DE6S
LOOP1:MOV P1,#0CH
    LCALL DE12S
    LCALL DX
    MOV P1,#21H
    LCALL DE12S
    LCALL NB
    LJMP LOOP1
DE12s: MOV R5,#120
    LJMP DE1
DE6s:  MOV R5,#60
    LJMP DE1

```

```

DE02s:  MOV R5,#02H
DE1:    MOV R6,#200
DE2:    MOV R7,#126
DE3:    DJNZ R7,DE3
        DJNZ R6,DE2
        DJNZ R5,DE1
        RET

DX:MOV R2,#10
LOOP2:MOV P1,#14H
        LCALL DE02S
        MOV P1,#04H
        LCALL DE02S
        DJNZ R2,LOOP2
        RET

NB:MOV R2,#10
LOOP3:MOV P1,#22H
        LCALL DE02S
        MOV P1,#20H
        LCALL DE02S
        DJNZ R2,LOOP3
        RET

INT0P:MOV P1,#24H
        LCALL DE6S
        RETI

```

C51 程序: e9-2.c

```

#include <reg51.h>
void delay10ms(int k)
{
    int m,n;
    while(k--)
    {
        for(m=0;m<10;m++)
        {
            for(n=0;n<125;n++);
        }
    }
}

/*void delay02s()
{
#pragma asm
    MOV R5,#2
DE1: MOV R6,#200
DE2: MOV R7,#126
DE3: DJNZ R7,DE3
    DJNZ R6,DE2
}
*/

```

```
        DJNZ    R5,DE1
#pragma endasm
}

void delay12s()
{
#pragma asm
        MOV R5,#120
DE4: MOV R6,#200
DE5: MOV R7,#126
DE6: DJNZ    R7,DE6
        DJNZR6,DE5
        DJNZ    R5,DE4
#pragma endasm
}
void delay6s()
{
#pragma asm
        MOV R5,#60
DE7: MOV R6,#200
DE8: MOV R7,#126
DE9: DJNZ    R7,DE9
        DJNZR6,DE8
        DJNZ    R5,DE7
#pragma endasm
}*/

void nbhs8c()
{
    int i;
    for(i=0;i<8;i++)
    {
        P1=0x22;//东西红灯亮,南北黄灯亮
        //delay02s();
        delay10ms(20);
        P1=0x20;//东西红灯亮,南北黄灯灭
        //delay02s();
        delay10ms(20);
    }
}

void dxhs8c()
{
    int j;
    for(j=0;j<8;j++)
    {
        P1=0x14;//东西黄灯亮,南北红灯亮
        //delay02s();
        delay10ms(20);
        P1=0x04;//东西黄灯灭,南北红灯亮
        //delay02s();
        delay10ms(20);
    }
}
```

```

    }
}
void main()
{
    P1=0x24; //东西南北红灯
    //delay6s();
    delay10ms(600);
    while(1)
    {
        P1=0x0c; //东西红灯，南北绿灯
        //delay12s();
        delay10ms(1200);
        dxhs8c();
        P1=0x21; //东西绿灯，南北红灯
        //delay12s();
        delay10ms(1200);
        nbhs8c();
    }
}

```

实验三 简单 I/O 口扩展实验

一、实验要求

以两个 74LS273 作为输出口，控制十二个发光二极管燃灭，模拟交通灯管理。

二、实验目的

- (1) 学习在单片机系统中扩展简单 I/O 接口的方法。
- (2) 学习数据输出程序的设计方法。
- (3) 学习模拟交通灯控制的实验方法。

三、实验说明

要完成本实验，首先必须了解交通路灯的燃灭规律。设有一个十字路口，2、4 为南北方向，1、3 为东西方向，初始状态为四个路口的红灯全亮。之后，东、西路口的绿灯亮，南、北路口的红灯亮，东、西路口方向通车，延时一段时间后，东西路口的绿灯灭，黄灯开始闪烁。闪烁若干次后，东、西路口的红灯亮，而同时南、北路口的绿灯亮，南、北路口方向通车。延时一段时间后，南北路口的绿灯灭，黄灯开始闪烁。闪烁若干次后，再切换到东、西路口方向。之后，重复上述过程。

各 LED 发光二极管共阳极，但各发光二极管阴极接有非门，因而使其点亮应使相应位置为高电平。

四、仿真实验电路

简单 I/O 口扩展实验仿真电路如图 9.4 所示。

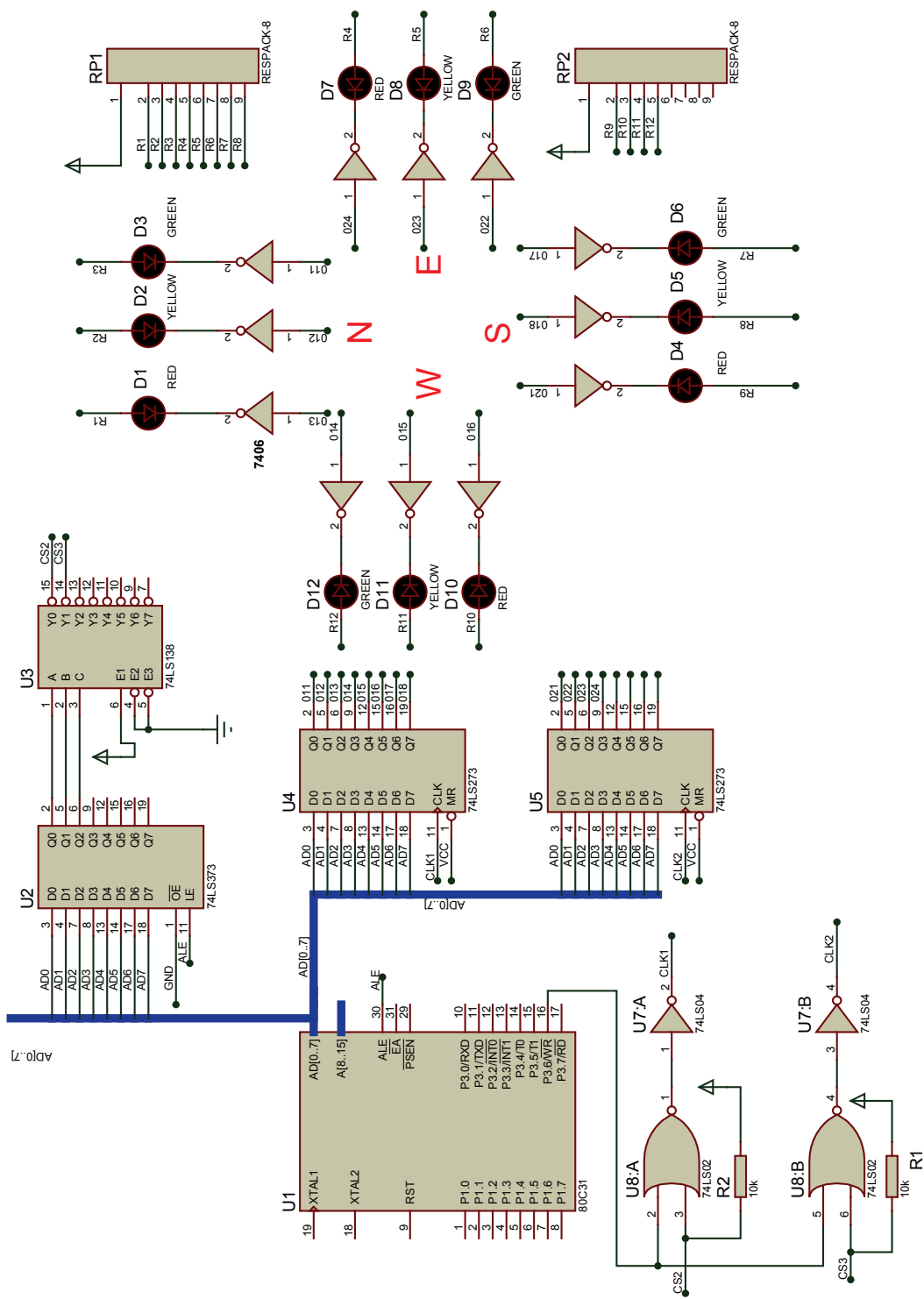


图 9.4 简单 I/O 口扩展实验仿真电路

五、参考程序清单

汇编语言程序：e9-3.asm

```
ORG 0000H
LJMP START
ORG 200H
START: MOV P2,#00H
MOV R0,#00H
MOV R1,#01H
MOV A,#24H
MOVX @R0,A
MOV A,#09H
MOVX @R1,A
LCALL DE6s
LLL: MOV A,#0CH
MOVX @R0,A
MOV A,#03H
MOVX @R1,A
LCALL DE12s
MOV A,#04H
MOVX @R0,A
MOV A,#01H
MOVX @R1,A
MOV R2,#08H
TTT: MOV A,#14H
MOVX @R0,A
MOV A,#05H
MOVX @R1,A
LCALL DE02s
MOV A,#04H
MOVX @R0,A
MOV A,#01H
MOVX @R1,A
LCALL DE02s
DJNZ R2,TTT
MOV A,#24H
MOVX @R0,A
MOV A,#09H
MOVX @R1,A
LCALL DE02s
MOV A,#61H
MOVX @R0,A
MOV A,#08H
MOVX @R1,A
LCALL DE12s
MOV A,#20H
MOVX @R0,A
MOV A,#08H
MOVX @R1,A
```



```

    MOV    R2,#08H
GGG: MOV    A,#0A2H
    MOVX   @R0,A
    LCALL  DE02s
    MOV    A,#20H
    MOVX   @R0,A
    LCALL  DE02s
    DJNZ   R2,GGG
    MOV    A,#24H
    MOVX   @R0,A
    MOV    A,#09H
    MOVX   @R1,A
    LCALL  DE02s
    JMP     LLL
DE12s: MOV    R5,#120
    LJMP   DE1
DE6s:  MOV    R5,#60
    LJMP   DE1
    DE02s: MOV    R5,#02H
DE1: MOV    R6,#200
DE2: MOV    R7,#126
DE3: DJNZ   R7,DE3
    DJNZ   R6,DE2
    DJNZ   R5,DE1
    RET
    END
C51 程序: e9-3.c
#include <reg51.h>
#include <absacc.h>
#define P273_1 XBYTE[0x0000]
#define P273_2 XBYTE[0x0001]

void delay100ms()
{
    #pragma asm
DE1:  MOV    R6,#200
DE2: MOV    R7,#126
DE3: DJNZ   R7,DE3
    DJNZ   R6,DE2
    DJNZ   R5,DE1
    RET
    #pragma endasm
}

void delay02s()
{
    #pragma asm
    MOV    R5,#2
    LCALL  DE1
    #pragma endasm

```

```
}

void delay12s()
{
#pragma asm
    MOV R5,#120
    LCALL DE1
#pragma endasm
}

void delay6s()
{
#pragma asm
    MOV R5,#60
    LCALL DE1
#pragma endasm
}

void dxhs8c()
{
int i;
for(i=0;i<8;i++)
{
    P273_1=0x14;    //东西黄灯亮，南北红灯亮
    P273_2=0x05;
    delay02s();
    P273_1=0x04;    //东西黄灯灭，南北红灯亮
    P273_2=0x01;
    delay02s();
}
}

void nbhs8c()
{
int j;
for(j=0;j<8;j++)
{
    P273_1=0xa2;    //东西红灯亮，南北黄灯亮
    P273_2=0x08;
    delay02s();
    P273_1=0x20;    //东西红灯亮，南北黄灯灭
    P273_2=0x08;
    delay02s();
}
}

void main()
{
    P273_1=0x24; //东西南北红灯
    P273_2=0x09;
    delay6s();
}
```

```
while(1)
{
    P273_1=0x0c; //东西绿灯，南北红灯
    P273_2=0x03;
    delay12s();
    dxhs8c();
    P273_1=0x61; //东西红灯，南北绿灯
    P273_2=0x08;
    delay12s();
    nbhs8c();
}
}
```

实验四 外部中断实验

一、实验要求

在上一实验（交通灯控制实验）内容的基础上增加允许急救车优先通过的要求。有急救车到达时，各方向交通灯信号为全红，以便让急救车通过，假定急救车通过路口的时间为 10 秒，急救车通过后，交通灯恢复中断前的状态。本实验以单脉冲为中断申请，表示有急救车通过。

二、实验目的

- (1) 学习外部中断技术的基本使用方法。
- (2) 学习中断处理程序的编程方法。

三、实验说明

交通灯的燃灭规律见实验三。

本实验中断处理程序的应用，最主要的地方是如何保护进入中断前的状态，使得中断程序执行完毕后能回到交通灯中断前的状态。要保护的地方除了累加器 ACC、PSW 外，还要注意：一是主程序中的延时程序和中断处理程序中延时程序不能混用。本实验中，主程序中的延时用的寄存器和中断延时用的寄存器应不相同。二是主程序中每执行一步经 74LS273 的端口输出数据操作时，要先将所输出的数据保存到一个单元中。因为进入中断程序后也要执行往 74LS273 端口输出数据的操作，中断返回时如果没有恢复中断前 74LS273 端口锁存器的数据，则显示往往出错，回不到中断前的状态。还要注意一点，主程序中往端口输出数据操作要先保存再输出。

后面给出的参考程序中没有保护现场，观察实验结果，然后修改程序。

四、仿真实验电路

外部中断实验仿真电路如图 9.5 所示。

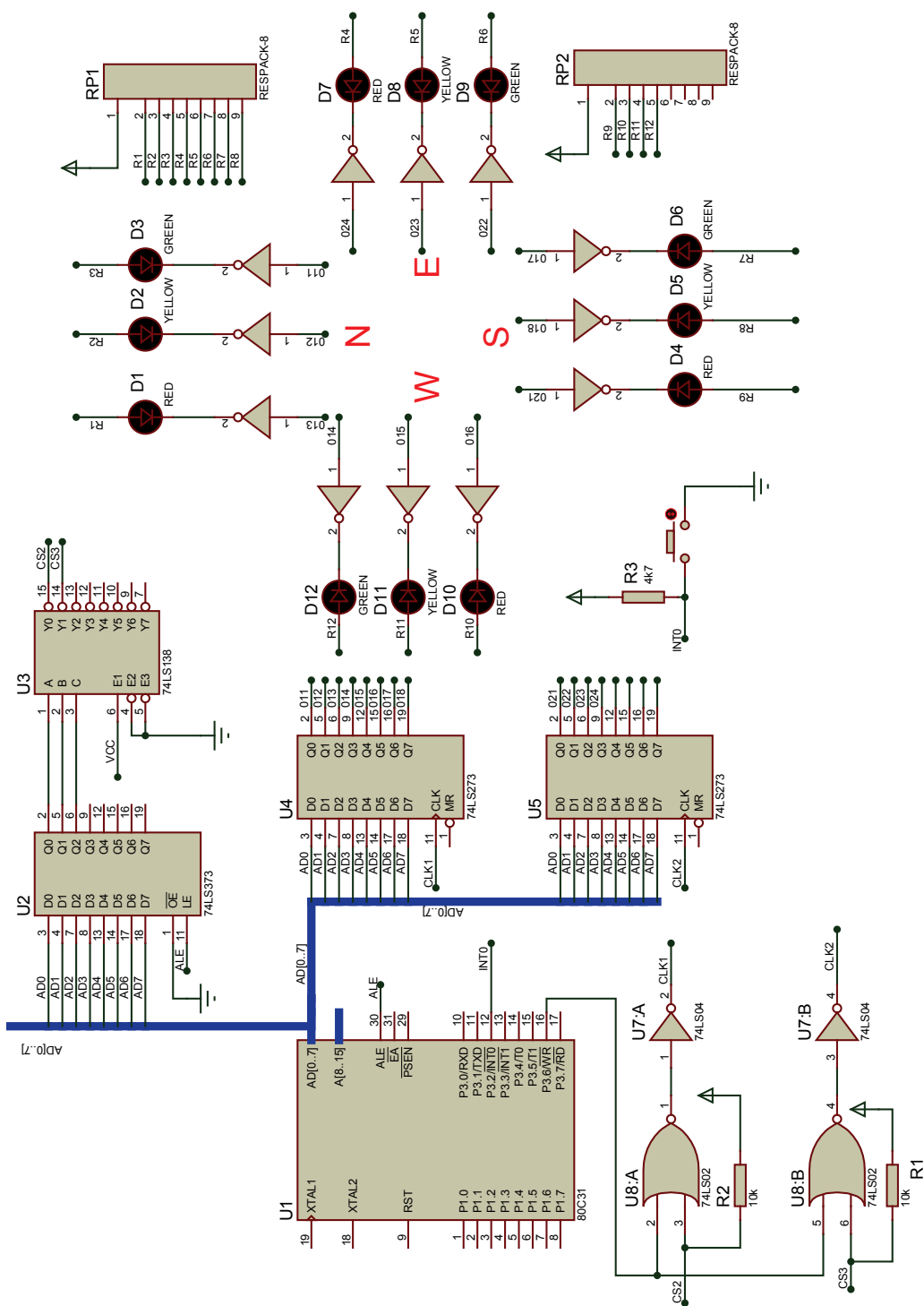


图 9.5 外部中断实验仿真电路

五、参考程序清单

```
#include <reg51.h>
#include <absacc.h>
#define P273_1 XBYTE[0x0000]
#define P273_2 XBYTE[0x0001]
void delay100ms()
{
    #pragma asm
DE1:    MOV  R6,#200
DE2: MOV R7,#126
DE3: DJNZ R7,DE3
        DJNZ R6,DE2
        DJNZ R5,DE1
        RET
    #pragma endasm
}

void delay02s()
{
    #pragma asm
        MOV R5,#2
        LCALL DE1
    #pragma endasm
}

void delay12s()
{
    #pragma asm
        MOV R5,#120
        LCALL DE1
    #pragma endasm
}

void delay6s()
{
    #pragma asm
        MOV R5,#60
        LCALL DE1
    #pragma endasm
}

void dxhs8c()
```

```
{
int i;
for(i=0;i<8;i++)
{
P273_1=0x14; //东西黄灯亮, 南北红灯亮
P273_2=0x05;
delay02s();
P273_1=0x04; //东西黄灯灭, 南北红灯亮
P273_2=0x01;
delay02s();
}
}

void nbhs8c()
{
int j;
for(j=0;j<8;j++)
{
P273_1=0xa2; //东西红灯亮, 南北黄灯亮
P273_2=0x08;
delay02s();
P273_1=0x20; //东西红灯亮, 南北黄灯灭
P273_2=0x08;
delay02s();
}
}

void dxallred() interrupt 0 using 1
{
P273_1=0x24; //东西南北红灯
P273_2=0x09;
delay6s();
}

void main()
{
EA=1;
EX0=1;
IT0=1;
P273_1=0x24; //东西南北红灯
P273_2=0x09;
delay6s();

while(1)
{
```

```

P273_1=0x0c; //东西绿灯，南北红灯
P273_2=0x03;
delay12s();
dxhs8c();
P273_1=0x61; //东西红灯，南北绿灯
P273_2=0x08;
delay12s();
nbhs8c();
}
}

```

实验五 定时器实验

一、实验要求

由 AT89C51 内部定时器 T1，按方式 1 工作，即作为 16 位定时器使用，每 0.1s，T1 溢出中断一次。P1 口的 P1.1~P1.7 分别接八个发光二极管。要求编写程序模拟一时序控制装置。开机后第 1s L1、L3 亮，第 2s L2、L4 亮，第 3s L5、L7 亮，第 4s L6、L8 亮，第 5s L1、L3、L5、L7 亮，第 6s L2、L4、L6、L8 亮，第 7s 八个二极管全亮，第 8s 八个二极管全灭，以后又从头开始，L1、L3 亮，然后 L2、L4 亮……一直循环下去。

二、实验目的

- (1) 学习 51 单片机内部定时/计数器的使用和编程方法。
- (2) 进一步掌握中断处理程序的编程方法。

三、实验说明

1. 定时常数的确定

定时器/计数器的输入脉冲周期与机器周期一样，为振荡器频率的 1/12。本实验中时钟频率为 6.144MHz，现要采用中断方法来实现 1s 延时，要在定时器 1 中设置一个时间常数，使其每隔 0.1s 产生一次中断，CPU 响应中断后 R0 中计数值减一，令 (R0)=0AH，即可实现 1s 延时。

时间常数可按下法确定：

机器周期=12÷6.144×10⁶=1.9531×10⁻⁶s，需设初值 X，则 (2¹⁶-X)×1.9531×10⁻⁶=0.1
X=14336

化为十六进制：X=3800H，故初始值 TH1=38H，TL1=00H

2. 初始化程序

包括定时器初始化和中断系统初始化，主要是对 IP、IE、TCON、TMOD 的相应位进

行正确的设置，并将时间常数送入定时器中。由于只有定时器中断，IP 不必设置。

注意一点，定时器 1 初始化时建议用下述指令：

`ANL TMOD, #0FH`

`ORL TMOD, #10H`

而不要用如下指令：

`MOV TMOD, #10H`

否则定时器 0 被屏蔽，可能会影响串行口波特率，使程序不能执行。

3. 设计中断服务程序和主程序

中断服务程序要将时间常数重新送入定时器中，为下一次中断做准备。主程序则用来控制发光二极管按要求顺序亮灭。

四、仿真实验电路

定时器实验仿真电路如图 9.6 所示。

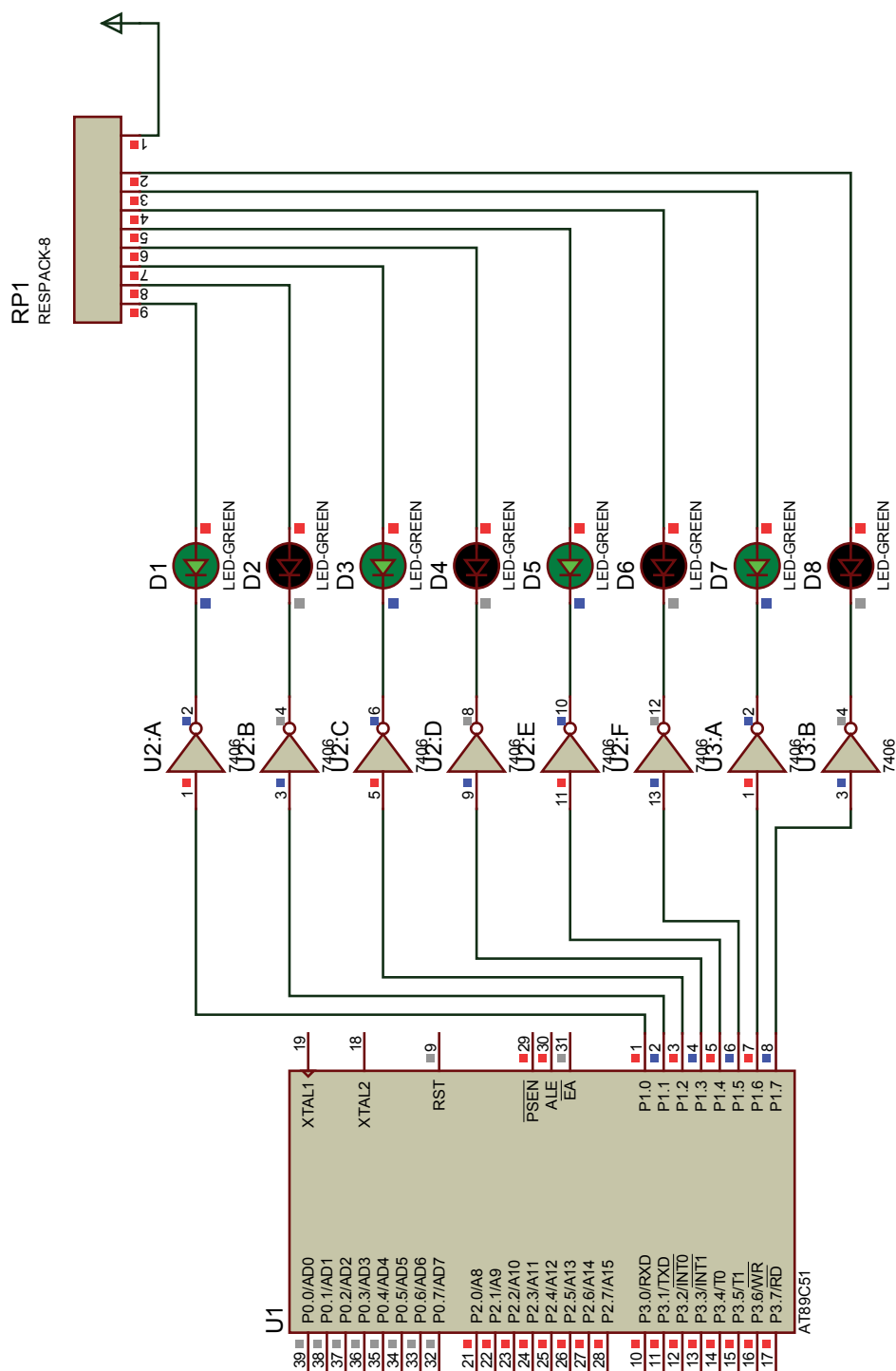


图 9.6 定时器实验仿真电路

五、参考程序清单

汇编语言程序: e9-5.asm

```

    ORG 0000H
    LJMP START
    ORG 000BH
    LJMP INT00
    ORG 200H
START:MOV A,#05H
      MOV DPTR,#TAB
      MOV R1,#0
      MOV R0,#0AH
      MOV TMOD,#01H
      MOV TH0,#3CH
      MOV TL0,0B0H
      MOV IE,#82H
      SETB TR0
    LOOP1:CJNE R0,#0,DISP
      MOV R0,#0AH
      INC R1
      CJNE R1,#8,LOOP2
      MOV R1,#0
    LOOP2:MOV A,R1
      MOVC A,@A+DPTR
      LJMP DISP
TAB:DB 05,0AH,50H,0A0H,55H,0AAH,0FFH,0
DISP:MOV P1,A
      LJMP LOOP1
INT00:CLR TR0
      DEC R0
      MOV TH0,#3CH
      MOV TL0,0B0H
      SETB TR0
      RETI
    END

```

C51 程序: e9-5.c

```

#include <reg51.h>
int a=0x0a;
int i=0;
int Tab[]={0x05,0x50,0x0a,0xa0,0x55,0xaa,0xff,0x00};
void main()
{
    TMOD=0x01;
    TH0=0x3c;
    TL0=0xb0;

```

```
EA=1;
ET0=1;
TR0=1;
while(1);
}
void int0() interrupt 1 using 0
{
    TR0=0;
    a=a-1;
    TH0=0x3c;
    TL0=0xb0;
    TR0=1;
    if(a!=0)
    {
        P1=Tab[i];
    }
    else
    {
        a=0x0a;
        i=i+1;
    }
    if(i==8) i=0;
}
```

实验六 8255A 可编程并行接口实验

一、实验要求

利用 8255A 可编程并行接口芯片，B 口作为输入口接 8 个开关，A 口作为输出口接 8 只发光二极管，开关控制相应发光二极管。

二、实验目的

- (1) 了解 8255A 芯片的结构及编程方法。
- (2) 掌握通过 8255A 读取开关数据的方法。

三、实验说明

先根据电路设计确定各端口的地址，向控制口写入方式控制字，设置好各端口的工作方式。实验中 A、B、C 都应工作在方式 0，B 口输入，A 口输出。方式控制字为 82H。

四、仿真实验电路

8255A 可编程并行接口实验仿真电路如图 9.7 所示。

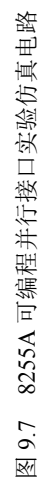


图 9.7 8255A 可编程并行接口实验仿真电路

五、参考程序清单

汇编语言程序：e9-9.asm

```
PA EQU 0FF20H;A 口地址
PB EQU 0FF21H;B 口地址
PC1 EQU 0FF22H;C 口地址
PK EQU 0FF23H;控制口地址
ORG 0000H
MOV DPTR,#PK;控制口地址
MOV A,#82H;送方式控制字
MOVX @DPTR,A
LOOP:MOV DPTR,#PB;B 口地址
MOVX A,@DPTR
MOV DPTR,#PA;A 口地址
MOVX @DPTR,A
LJMP LOOP
```

C51 程序：e9-9.c

```
#include<reg51.h>
#include<absacc.h>
#define PA XBYTE[0xff20]
#define PB XBYTE[0xff21]
#define PK XBYTE[0xff23]
void main()
{
    int a;
    PK=0x82;//初始化
    while(1)
    {
        a=PB;
        PA=a;
    }
}
```

实验七 数码显示实验

一、实验要求

用共阳极 7 段显示器显示按键的键值,要求用 74LS164 作为显示接口,用矩阵式键盘,至少识别 16 个按键,键值能在 7 段显示器上显示。

二、实验目的

- (1) 理解 LED 7 段数码管的显示控制原理。
- (2) 掌握数码管与单片机的接口技术,能够编写数码管显示驱动程序。

(3) 熟悉接口程序调试方法。

三、实验说明

用 74LS164 作为显示接口，单片机串行口工作在方式 0。可将共阳极 7 段显示器的段码放在数组中，根据键值查数组，通过串口送出段码并显示。

四、仿真实验电路

数码显示实验仿真电路如图 9.8 所示。

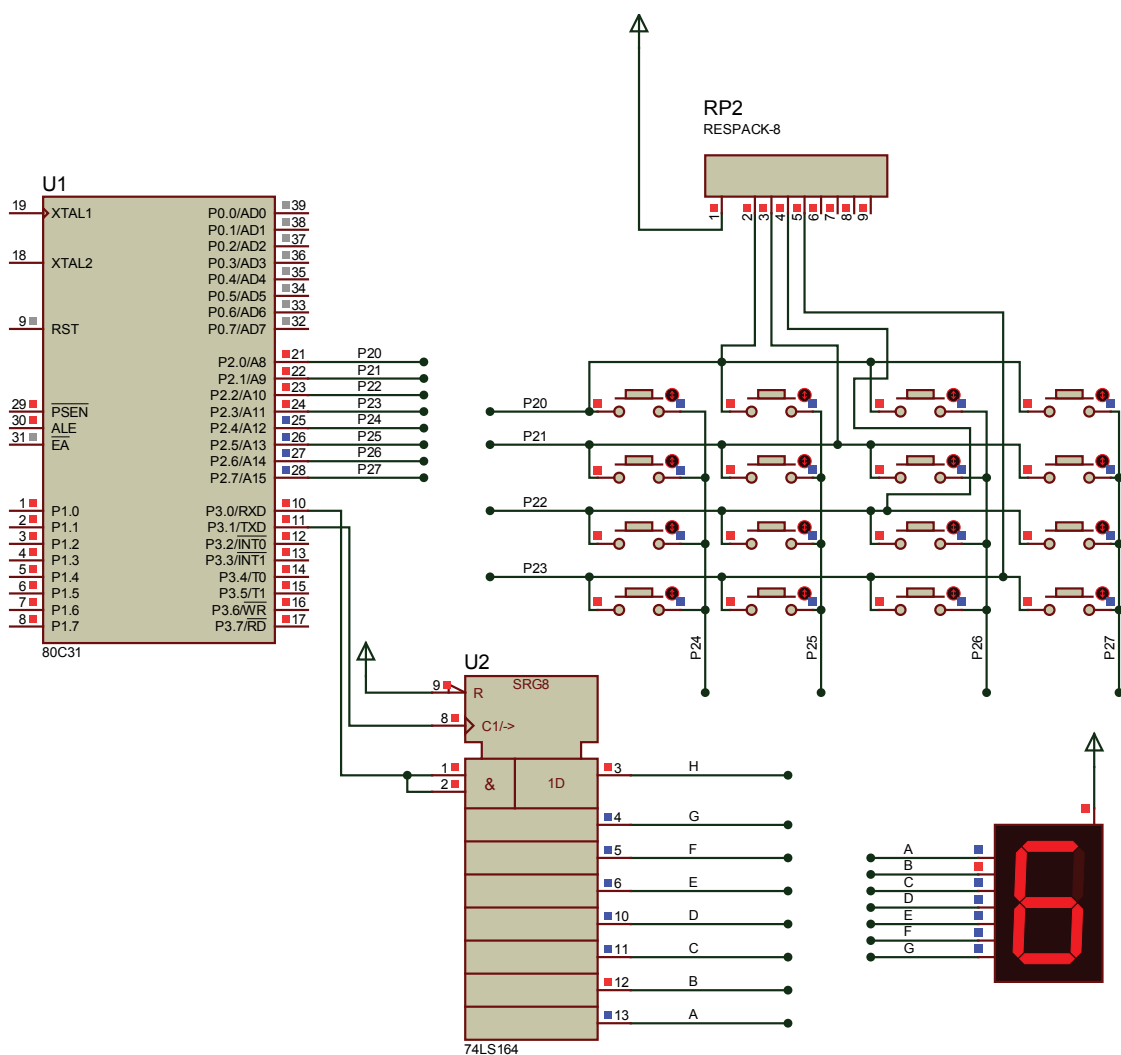


图 9.8 数码显示实验仿真电路

五、参考程序清单

```

#include <REG51.h> // MCS-51 头文件声明
#define uchar unsigned char // 定义无符号变量简写式
#define uint unsigned int //
char led[]={0xC0,0xF9,0xa4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,
0xC6,0xA1,0x86,0x8E};
sfr KEY=0xa0; // 定义 P2 口为键盘接口
void Delay(uchar); // 延时程序声明 (单位为毫秒)
uchar key();
uchar Key() // 键盘处理函数
{
    uchar a,b,c; // 定义 3 个变量
    KEY=0x0f; // 键盘口置 00001111
    if (KEY!=0x0f) // 查寻键盘口的值是否变化
    {
        Delay(20); // 延时 20 毫秒
        if (KEY!=0x0f) // 有键按下处理
        {
            a=KEY; // 键值放入寄存器 a
        }
        KEY=0xf0; // 将键盘口置为 11110000
        c=KEY; // 将第二次取得值放入寄存器 c
        a=a|c; // 将两个数据融合
        switch(a) // 对比数据值
        {
            case 0xee: b = 0x00; break; // 对比得到的键值给 b 一个应用数据
            case 0xde: b = 0x01; break;
            case 0xbe: b = 0x02; break;
            case 0x7e: b = 0x03; break;
            case 0xed: b = 0x04; break;
            case 0xdd: b = 0x05; break;
            case 0xbd: b = 0x06; break;
            case 0x7d: b = 0x07; break;
            case 0xeb: b = 0x08; break;
            case 0xdb: b = 0x09; break;
            case 0xbb: b = 0x0a; break;
            case 0x7b: b = 0x0b; break;
            case 0xe7: b = 0x0c; break;
            case 0xd7: b = 0x0d; break;
            case 0xb7: b = 0x0e; break;
            case 0x77: b = 0x0f; break;
            default: break; // 键值错误处理
        }
    }
    return(b); // 将 b 作为返回值
}

```

```
void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //一个 ; 表示空语句,CPU 空转。
        //i 从 0 加到 125, CPU 大概就耗时 1 毫秒
    }
}

void main()
{
    SCON=0x00;
    while(1)
    {
        SBUF=led[key()];
        while(TI==0);
        TI=0;
        Delay(5);
    }
}
```

实验八 液晶显示屏 1602 显示实验

一、实验题目

用 Protues 设计一 LCM1602 液晶显示接口电路。要求利用 P0 口接 LCM1602 液晶的数据口, P2.0~P2.2 做 LCM1602 液晶的控制信号输入端。编写程序, 实现字符串的显示。

二、实验目的

- (1) 掌握 LCM1602 液晶模块显示西文的原理及使用方法。
- (2) 掌握 8 位数据模式驱动 LCM1602 液晶的 C 语言编程方法。
- (3) 掌握用 LCM1602 液晶模块显示字符的 C 语言编程方法。

三、实验说明

液晶显示屏(LCD,Liquid Crystal Display)主要用于显示文本及图形信息。液晶显示屏具有轻薄、体积小、耗电量低、无辐射危险、平面直角显示, 以及影像稳定不闪烁等特点。因此, 在许多电子应用系统中, 常使用液晶显示屏作为人机界面。本实验采用的 1602 液晶模块是 2 行 16 个字的显示模块, 其内部有 80×8 位的 RAM 数据缓冲区。

四、仿真实验电路

液晶显示屏 1602 显示实验仿真电路如图 9.9 所示。

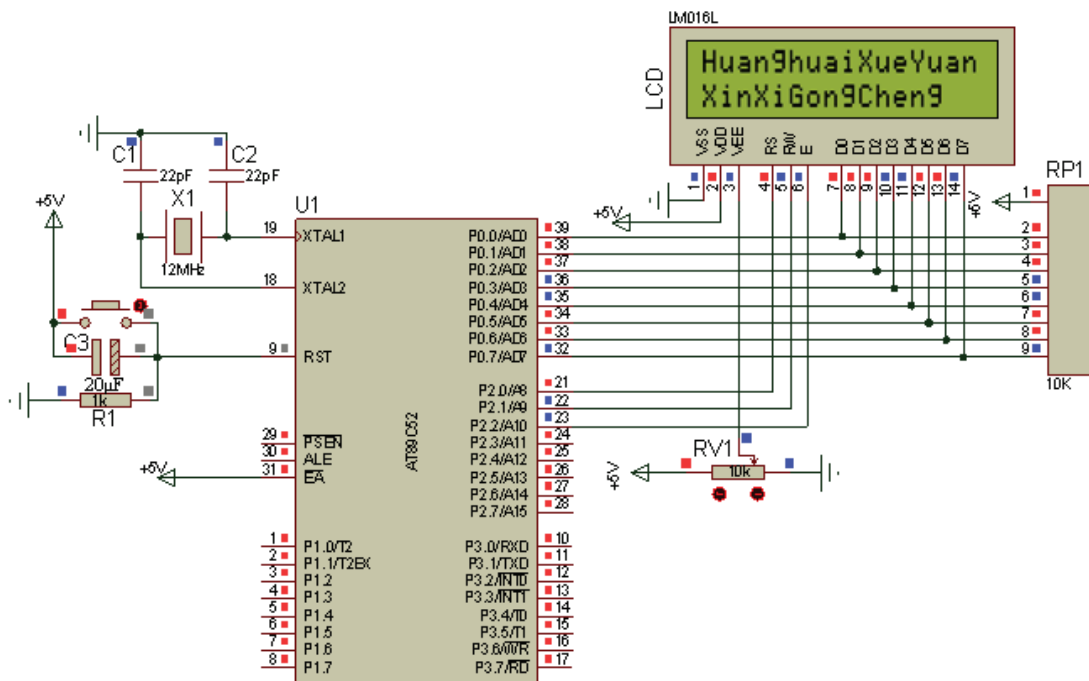


图 9.9 液晶显示屏 1602 显示实验仿真电路

五、参考程序清单

```

/*****包含头文件*****/
#include <reg51.h>
#include <intrins.h>
/*****端口定义*****/
sbit rs=P2^0;
sbit rw=P2^1;
sbit e=P2^2;
/*****
函数功能:LCD 延时子程序
入口参数:ms
出口参数:
*****/
void delay(unsigned char ms)
{
    unsigned char i;
    while(ms--)
    {
        for(i = 0; i < 250; i++)
        {
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
    }
}

```

```

}
}
/*****
函数功能:测试 LCD 忙状态
入口参数:
出口参数:result
*****/
bit lcd_bz()
{
bit result;
rs = 0;
rw = 1;
e = 1;
_nop_();
_nop_();
_nop_();
_nop_();
result = (bit)(P0 & 0x80);
e = 0;
return result;
}
/*****
函数功能:写指令数据到 LCD 子程序
入口参数:cmd
出口参数:
*****/
void lcd_wcmd(unsigned char cmd)
{
while(lcd_bz()); //判断 LCD 是否忙
rs = 0;
rw = 0;
e = 0;
_nop_();
_nop_();
P0 = cmd;
_nop_();
_nop_();
_nop_();
_nop_();
e = 1;
_nop_();
_nop_();
_nop_();
_nop_();
e = 0;
}
/*****
函数功能:设定显示位置子程序
入口参数:pos
出口参数:

```

```

*****/
void lcd_pos(unsigned char pos)
{
    lcd_wcmd(pos | 0x80);
}
/*****
函数功能:写入显示数据到LCD子程序
入口参数:dat
出口参数:
*****/
void lcd_wdat(unsigned char dat)
{
    while(lcd_bz()); //判断LCD是否忙
    rs = 1;
    rw = 0;
    e = 0;
    P0 = dat;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    e = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    e = 0;
}
/*****
函数功能:LCD初始化子程序
入口参数:
出口参数:
*****/
void lcd_init()
{
    lcd_wcmd(0x38);
    delay(1);
    lcd_wcmd(0x0c);
    delay(1);
    lcd_wcmd(0x06);
    delay(1);
    lcd_wcmd(0x01);
    delay(1);
}
/*****显示数据表*****/
unsigned char code dis1[] = {"HuanghuaiXueYuan"};
unsigned char code dis2[] = {"XinXiGongCheng"};
/*****
函数功能:主程序
入口参数:

```

出口参数:

```
*****/  
void main(void)  
{  
    unsigned char i;  
    lcd_init();// 初始化 LCD  
    delay(10);  
    lcd_pos(0x00);//设置显示位置  
    i = 0;  
    while(dis1[i] != '\0')  
    {  
        lcd_wdat(dis1[i]);//显示字符  
        i++;  
    }  
    lcd_pos(0x40);// 设置显示位置  
    i = 0;  
    while(dis2[i] != '\0')  
    {  
        lcd_wdat(dis2[i]);// 显示字符  
        i++;  
    }  
    while(1);  
}
```

实验九 串/并转换实验

一、实验要求

利用单片串行口和串入并出移位寄存器 74LS164 扩展输出口,在数码显示器上循环显示 0~9 这 10 个数字。

二、实验目的

- (1) 掌握单片机串行口工作方式及编程方法。
- (2) 掌握利用串行口扩展 I/O 口的方法。

三、实验说明

串行口工作在方式 0 时,可通过外接移位寄存器实现串并转换。在这种方式下,数据为 8 位,从 RXD 端输入/输出,TXD 端用于输出移位同步时钟信号,波特率为 $f_{osc}/12$ 。编程时,先由软件设置 SCON,在 CPU 将数据写入 SBUF 后立即启动发送,待 8 位数据接收完后硬件将 TI 置 1,必须由软件将其清零。

四、仿真实验电路

串/并转换实验仿真电路如图 9.10 所示。

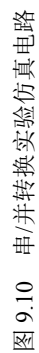


图 9.10 串/并转换实验仿真电路

五、参考程序清单

汇编语言程序: e9-9.asm

```

ORG 0000H
    LJMP START
    ORG 0200H
START:MOV R4,#00H
    MOV SCON,#00H
LOOP:MOV DPTR,#SEG
    MOV A,R4
    MOVC A,@A+DPTR
    MOV SBUF,A
    JNB TI,$
    CLR TI
    LCALL DEL
    INC R4
    CJNE R4,#0AH,LOOP
    MOV R4,#00H
    SJMP LOOP

DEL:    MOV    R1,#10
DEL1:   MOV    R2,#200
DEL2:   MOV    R3,#126
DEL3:   DJNZ   R3,DEL3
        DJNZ   R2,DEL2
        DJNZ   R1,DEL1

    RET
SEG:DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH
END

```

C51 程序: e9-9.c

```

#include <reg52.h>
unsigned char code dispcode[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
0x7F,0x6F};
void delay(void)
{
    int m,n;
    for(m=0;m<400;m++)
        for(n=0;n<400;n++)
            ;
}
void main()
{
    int i=0;
    SCON=0x00;

    for(i=0;i<10;i++)
    {
        SBUF=dispcode[i];
    }
}

```

```

delay();
while(TI==0)
;
TI=0;
if(i==10)
{
i=0;
}
}
}

```

实验十 A/D 转换实验

一、实验要求

利用 ADC0808(ADC0809), 由电位器提供模拟量输入, 编写程序, 将模拟量转换成数字字量, 用 7 段显示器显示。

二、实验目的

- (1) 掌握 A/D 转换器与单片机的接口方法。
- (2) 了解 ADC0808(ADC0809)转换性能及编程方法。
- (3) 通过实验了解数据采集的方法。

三、实验说明

A/D 转换器大致有三类: 一是双积分 A/D 转换器, 优点是精度高, 抗干扰性好, 价格便宜, 但速度慢; 二是逐次逼近法 A/D 转换器, 精度、速度、价格适中; 三是并行 A/D 转换器, 速度快, 价格也昂贵。

实验用的 ADC0809 属第二类, 是八位 A/D 转换器。每采集一次需 100 μ s。

ADC0809 START 端为 A/D 转换启动信号, ALE 端为通道选择地址的锁存信号。实验电路中将其相连, 以便同时锁存通道地址并开始 A/D 采样转换, 故启动 A/D 转换器只需如下两条指令:

```

MOV    DPTR, #PORT
MOVX   @DPTR,A

```

A 中为何内容并不重要, 这是一次虚拟写。

在中断方式下, A/D 转换结束后会自动产生 EOC 信号, 将其与 INT0 相连接。在中断处理程序中, 使用如下指令即可读取 A/D 转换的结果:

```

MOV    DPTR, #PORT
MOVX   A, @DPTR

```

四、仿真实验电路

A/D 转换实验仿真电路如图 9.11 所示。

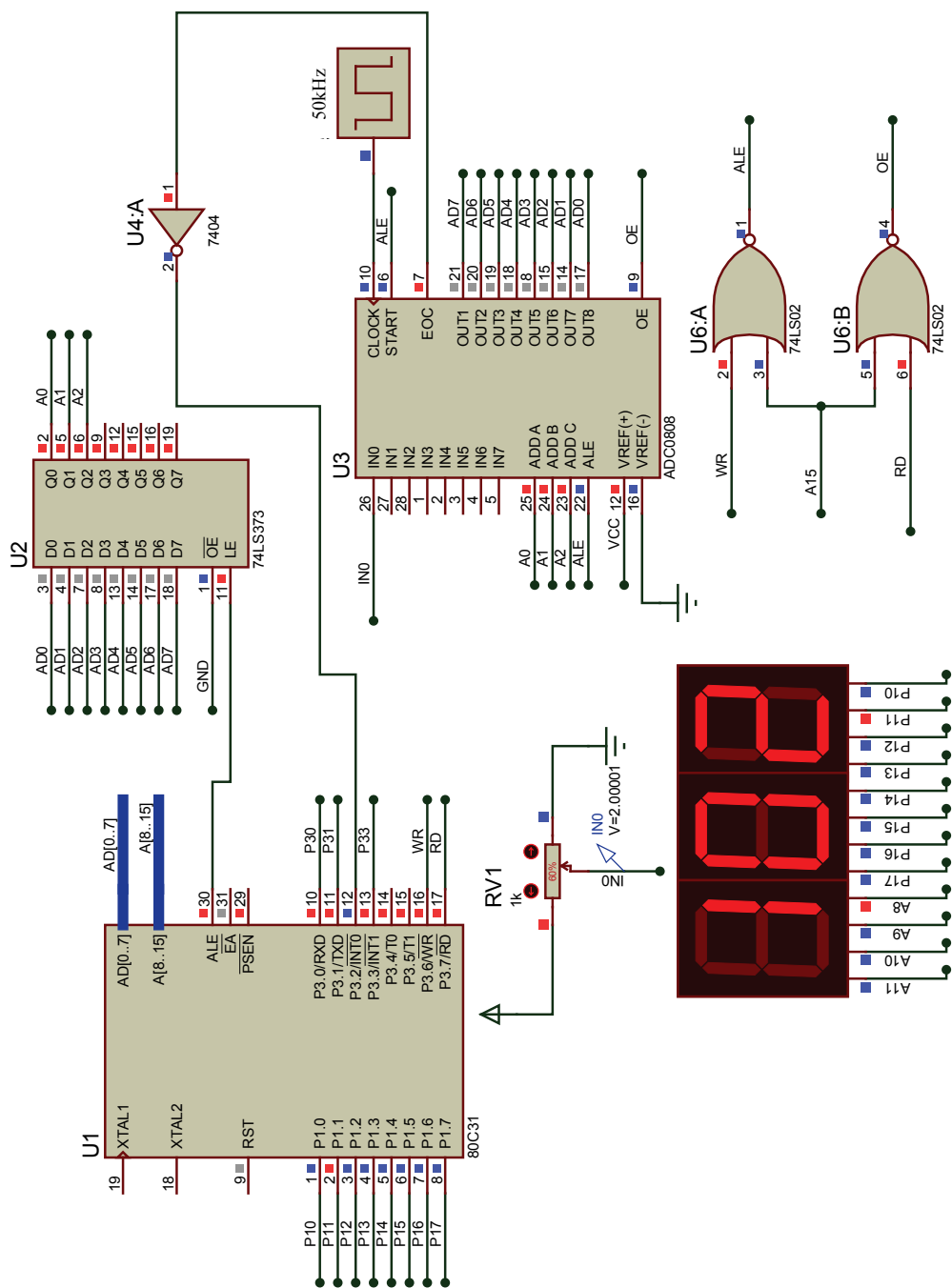


图 9.11 A/D 转换实验仿真电路

五、参考程序清单

汇编语言: ADC0808.asm

```

PORT EQU 7FF8H;通道 0 地址
    ORG    0000H
    LJMP   MAIN
    ORG    0200H
MAIN:    MOV DPTR,#PORT
LOOP:    MOVX @DPTR,A;启动转换
        MOV    R7,#24H;延时 100us 左右
DELAY:   NOP
        NOP
        NOP
        NOP
        NOP
        DJNZ  R7,DELAY
        MOVX  A,@DPTR;读结果
        MOV   R2,A
        LCALL B8TOBCD;调用将 8 位二进制数转换成 BCD 码子程序
        MOV  P1,R5
        MOV  P2,R6
        SJMP LOOP

;入口参数 R2, 出口参数 R6 (百位), R5 (十位个位)
B8TOBCD: MOV  A,R2
        MOV  B,#100
        DIV  AB
        MOV  R6,A
        MOV  A,#10
        XCH  A,B
        DIV  AB
        SWAP A
        ADD  A,B
        MOV  R5,A
        RET

END

```

C51 程序: adc0808-cx.c

```

#include<reg51.h>
#include<absacc.h>
#define ADC0808 XBYTE[0x7ff8]
sbit EOC=P3^2;
void delay(unsigned char j)
{
    while(j--);
}
void main(void)
{

```

```
unsigned char a,bai,shi,ge;
ADC0808=0;//启动转换
while(1)
{
    if(!EOC)
    {
        a=ADC0808;//读结果
        bai=a/100;
        shi=a%100/10;
        ge=a%10;
        P2=bai;
        P1=shi<<4|ge;
        ADC0808=0;//启动转换
    }
    delay(200);
}
```

C51 程序: adc0808-int.c

```
#include<reg51.h>
#include<absacc.h>
#define ADC0808 XBYTE[0x7ff8]
sbit EOC=P3^2;
unsigned char a;
void delay(unsigned char j)
{
    while(j--);
}
void main(void)
{
    EA=1;
    EX0=1;
    IT0=1;
    while(1)
    {
        ADC0808=0;
        delay(200);
        P2=a%1000/100;
        P1=((a%100/10)<<4)|(a%10);
        ADC0808=0;
    }
}

void int00() interrupt 0 using 1
{
    a=ADC0808;
}
```



第 10 章 单片机课程设计实例

实例一 基于单片机的简易计算器设计

一、设计要求

使用 1602C 字符显示液晶和 4×4 矩阵键盘设计一个简易计算器。

二、实例分析

由于使用 4×4 矩阵键盘只有 16 个按键，所以在设计中这样来设计按键：0~9 数字键，加、减、乘、除、等号，还有清零键，正好 16 个键。

三、硬件仿真电路

基于单片机的简易计算器设计仿真电路如图 10.1 所示。

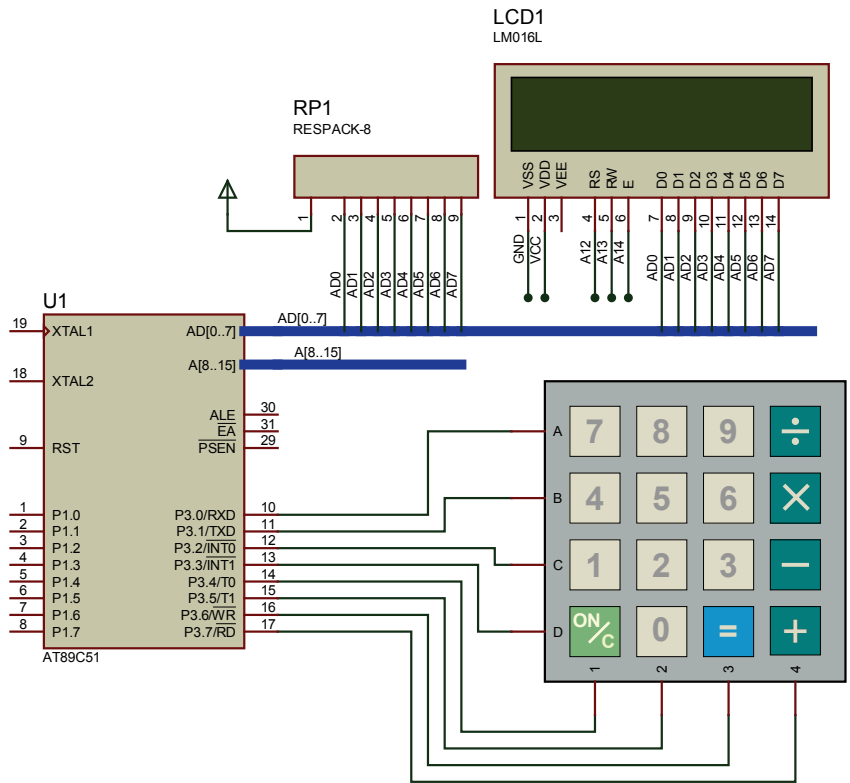


图 10.1 基于单片机的简易计算器设计仿真电路

四、程序设计

程序如下：

```
/* Lcd.h 包含 1602C 液晶的处理的头文件*/
#define uchar unsigned char
sbit RS=P2^4;
sbit RW=P2^5;
sbit E=P2^6;
void delay()
{
    uchar i=200;
    while(i--);
}
void fbusy()                /*忙函数*/
{
    P0=0xff;
    RS=0;
    RW=1;
    E=1;
    while(P0&0x80);
    E=0;
}
void writecmd(uchar cmd)    /*写一个字节命令*/
{
    fbusy();                /*检查忙*/
    RS=0;
    RW=0;
    E=1;
    P0=cmd;
    E=0;
}
void writedata(uchar date) /*写一个字节数据*/
{
    fbusy();
    E=1;
    RS=1;
    RW=0;
    P0=date;
    E=0;
    delay();
}

void init()                /*初始化*/
{
    writecmd(0x01);         /*清屏*/
    writecmd(0x38);         /*使用 8 位数据，显示两行，使用 5×7 的字型*/
}
```

```

    writecmd(0x0f);          /*显示器开, 光标开, 字符不闪烁 */
    writecmd(0x06);          /*字符不动, 光标自动右移动一格*/
    writecmd(0x80);
}
/* 键盘扫描及计算头文件*/
#include <stdio.h>
uchar date[15];
float num1=0;    /*存储第一个操作数*/
float num2=0;    /*存储第二个操作数*/
char opeart='#'; /*存储运算符*/
float zhi=0;      /*存储运算结果*/

void delay1()      /*延时, 主要用于消除抖动*/
{
    int i=20000;
    while(i--);
}
uchar keyscanx()    /*键盘行扫描*/
{
    uchar i;
    uchar keyx=0x01;
    for(i=0;i<4;i++)
    {
        if((P3&keyx)==0)
            break;
        else
            keyx=keyx<<1;
    }
    return i*4;
}

uchar keyscany()    /*键盘列扫描 8*/
{
    uchar i;
    P3=0x7f;
    for(i=0;i<4;i++)
    {
        if((P3&0x0f)!=0x0f)
            break;
        else
            P3=P3>>1|0x80;
    }
    return i;
}

uchar KeyScan()
{
    return keyscanx()+keyscany(); /*返回扫描的键值*/
}

```

```

}

void JudgeKey()
{
    uchar key;
    uchar i;
    P3=0x0f;
    if((P3&0x0f)!=0x0f)                /*判断是否有键按下
    {
        delay();                        /*去抖
        if((P3&0x0f)!=0x0f)
        {
            key=KeyScan();
            if(num1==0&&(key==0||(key>=11&&key<=15)));
            else
            {
                if(key>=0&&key<=9)
                {
                    if(num1==0&&key==0);
                    else
                        writedata(key+'0');
                    if(opeart=='#')        /*如果运算符为‘#’，初始化第一个数
                    {
                        num1=num1*10+key;
                    }
                    else                    /*否则处理第二个数
                    {
                        num2=num2*10+key;
                    }
                }

                if(key==11&&num1!=0&&num2!=0)
                {
                    writedata('=');
                    switch(opeart)
                    {
                        case '+': zhi=num1+num2;break;
                        case '-': zhi=num1-num2;break;
                        case '*': zhi=num1*num2;break;
                        case '/': zhi=num1/num2;break;
                    }
                    sprintf(date,"%0.5f",zhi);        /*库函数，float 类型的
数据转换成 char 类型的数组
                    for(i=0;date[i]!='\0';i++)
                        writedata(date[i]);
                    num1=0;
                    num2=0;
                    opeart='#';

```

```

        writecmd(0x80+0x40);
    }
    if(opeart!='#');
    else
    {
        switch (key)
        {
            case 12:opeart='+';writedate(opeart);break;
            case 13:opeart='-';writedate(opeart);break;
            case 14:opeart='*';writedate(opeart);break;
            case 15:opeart='/';writedate(opeart);break;
            default:break;
        }
    }
    if(key==10)
    {
        writecmd(0x01);
        num1=0;
        num2=0;
        opeart='#';
    }
}
}
}
/* 主程序*/
#include <reg52.h>
#include "Lcd.h"
#include "KeyScan.h"
void main()
{
    init();
    while(1)
    {
        JudgeKey();
        delay1();
    }
}

```

五、实例小结

本设计是以 AT89C51 单片机为核心的简易计算器设计，输入采用 4×4 矩阵键盘，可以进行加、减、乘、除带符号数字运算（八位整数），并在液晶显示屏 LCD1602 上静态显示操作的过程及结果。软件用 C 语言编程、Keil uVision2 和 pretues 仿真。

实例二 基于单片机的数字电压表设计

一、设计要求

利用 ADC0809(ADC0808)设计一个 5V 直流数字电压表。要求将输入的直流电压转换成数字信号，并将对应的电压值在 1602LCD 上显示出来。

二、实例分析

先输入 3 位通道选择地址，使 $ALE=1$ ，地址经译码选通某一路模拟输入到比较器，START 上升沿，逐次逼近寄存器复位。START 下降沿启动 A/D 转换，EOC 变低电平，直到转换完成，EOC 变为高电平，表示 A/D 转换结束，转换结果的数字量已存入锁存器，当 OE 输入为高电平时，输出三态门打开，数字量输出到数据总线上。由单片机进行处理后将对应的电压值显示在 LCD 上。

三、硬件仿真电路

基于单片机的数字电压表设计仿真电路如图 10.2 所示。

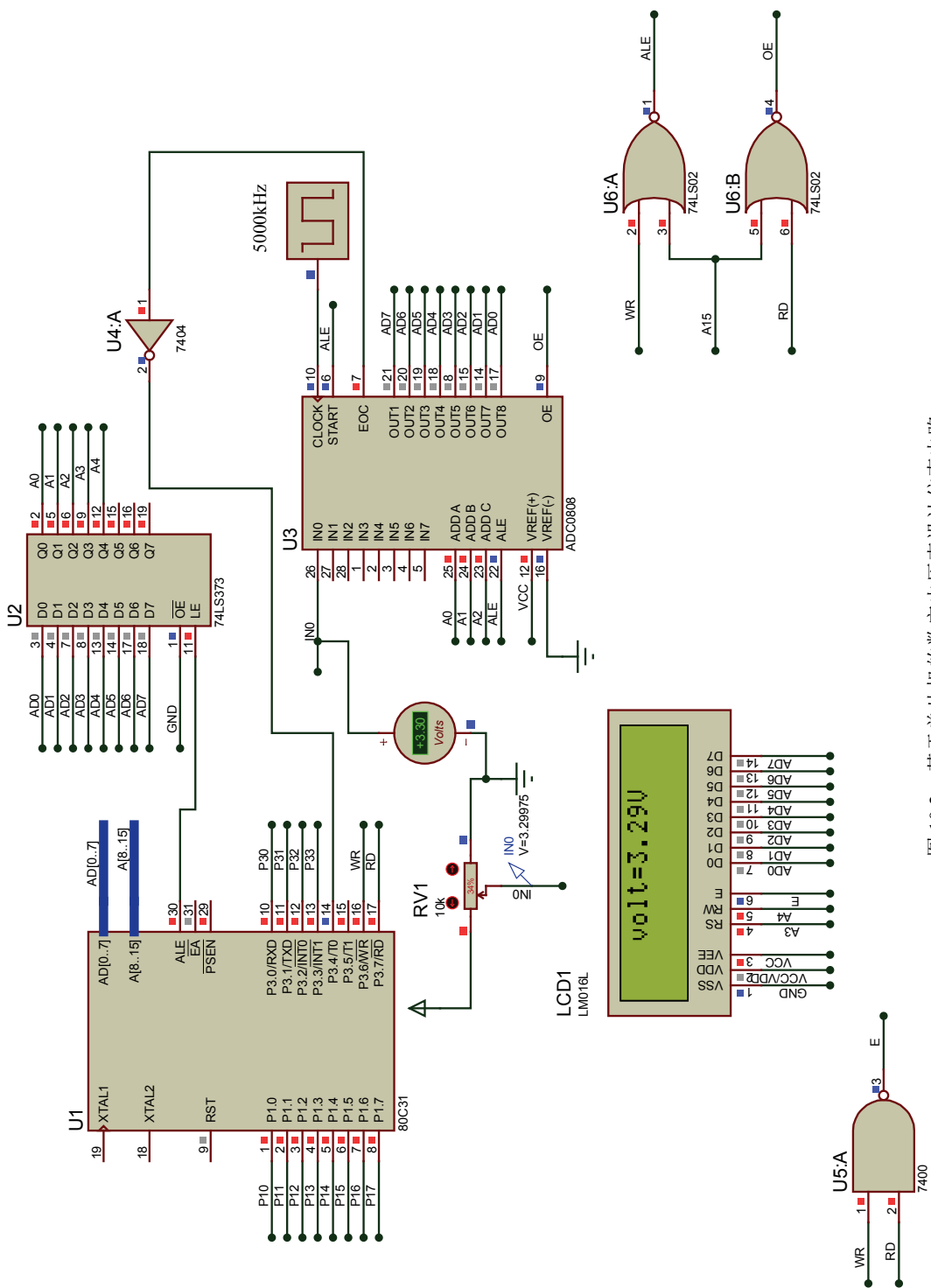


图 10.2 基于单片机的数字电压设计仿真电路

四、程序设计

```

***0808-lcd.c***
#include<reg51.h>
#include<absacc.h>
#include<lcd.h> //LCD 初始化
#define ADC0808 XBYTE[0x7ff8] //通道 0 地址
sbit EOC=P3^4;
void delay1(unsigned char j)
{
    while(j--);
}
void main(void)
{
    unsigned char i,a,zh,xiao;
    ADC0808=0; //启动转换
    while(1)
    {
        if(!EOC) //判断是否转换结束
        {
            lcdini();
            while(1)
            {
                a=ADC0808; //读结果
                zh=a/51; //求电压值整数
                lcd[5]=zh+0x30; //将电压整数部分转换成 ASCII 码写入数组
                xiao=(a%51)*100/51; //电压小数处理
                lcd[7]=(xiao/10)+0x30; //小数点后的第 1 位 ASCII 码写入数组
                lcd[8]=(xiao%10)+0x30; //小数点后的第 2 位 ASCII 码写入数组
                busy(); //查忙闲
                CMD_WR=0x83; //LCD 显示地址
                for(i=0;i<10;i++)
                {
                    busy(); //查忙闲
                    DATA_WR=lcd[i]; //写显示数据
                }
            }
            ADC0808=0; //再次启动转换
        }
        delay1(200);
    }
}

***lcd.h***
#define CMD_WR XBYTE[0x00]
#define DATA_WR XBYTE[0x08]
#define BUSY_RD XBYTE[0x10]
#define DATA_RD XBYTE[0x18]

```

```
#define CLS 0x01
#define HOME 0x02
#define SETMODE 0x06
#define SETVISIBLE 0x08
#define SHIFT 0x10
#define SETFUNCTION 0x38
#define SETCGADDR 0x40
#define SETDDADDR 0x80
char lcd[]={"volt=1.34V6789ABCDEF"};
void busy(void);
void Delay(unsigned char j);
void lcdini(void)
{
    Delay(20);
    busy();
    CMD_WR=SETFUNCTION;
    busy();
    CMD_WR=SETVISIBLE;
    busy();
    CMD_WR=CLS;
    busy();
    CMD_WR=SETMODE;
    busy();
    CMD_WR=0x0c;
}
void busy(void)
{
    Delay(5);
}

void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //1 毫秒延时
    }
}
```

五、实例小结

ADC0809 与 51 单片机的接口有以下 3 种方式：

查询方式、中断方式和延时方式，本例设计采用查询方式。

实例三 基于单片机的电子日历设计

一、设计要求

要求用 AT89C51 作为主控制器，用 8279 作为键盘和显示接口设计一个电子日历和电子钟，用 LED 分别显示当前时间和日期，具备校正日历和时间的功能。

电子日历做成如下两种格式（按 F 键两种格式可以切换）：

XX-XX-XX 由左向右分别为：时、分、秒

XX-XX-XX 由左向右分别为：年、月、日

C 键：清除，显示 00-00-00。

A 键：启动，电子钟（日历）计时。

D 键：停止，电子钟（日历）停止计时。

B 键：设置初值：由左向右依次输入预置的时、分、秒（年、月、日）值，同时应具有判断输入错误的能力，若输入有错，则显示：00-00-00 按 B 键即可重新输入预置值。

F 键：实现电子钟和日历的切换。

E 键：程序退出。

二、实例分析

用 AT89C51 作为主控制器，用 8279 作为键盘显示接口，控制键盘、LED 数码管等的操作和显示。

8279 主要特点如下：

- (1) 可同时进行键盘扫描及文字显示；
- (2) 键盘扫描模式（Scanned Keyboard Mode）；
- (3) 传感器扫描模式（Scanned Sensor Mode）；
- (4) 激发输入模式（Strobe Input Entry Mode）；
- (5) 8 乘 8 键盘 FIFO（先进先出）；
- (6) 具有接点消除抖动，2 键锁定及 N 键依此读出模式；
- (7) 双排 8 位数或双排 16 位数的显示器；
- (8) 右边进入或左边进入 16 位字节显示存储器。

三、硬件仿真电路

基于单片机的电子日历设计仿真电路如图 10.3 所示。

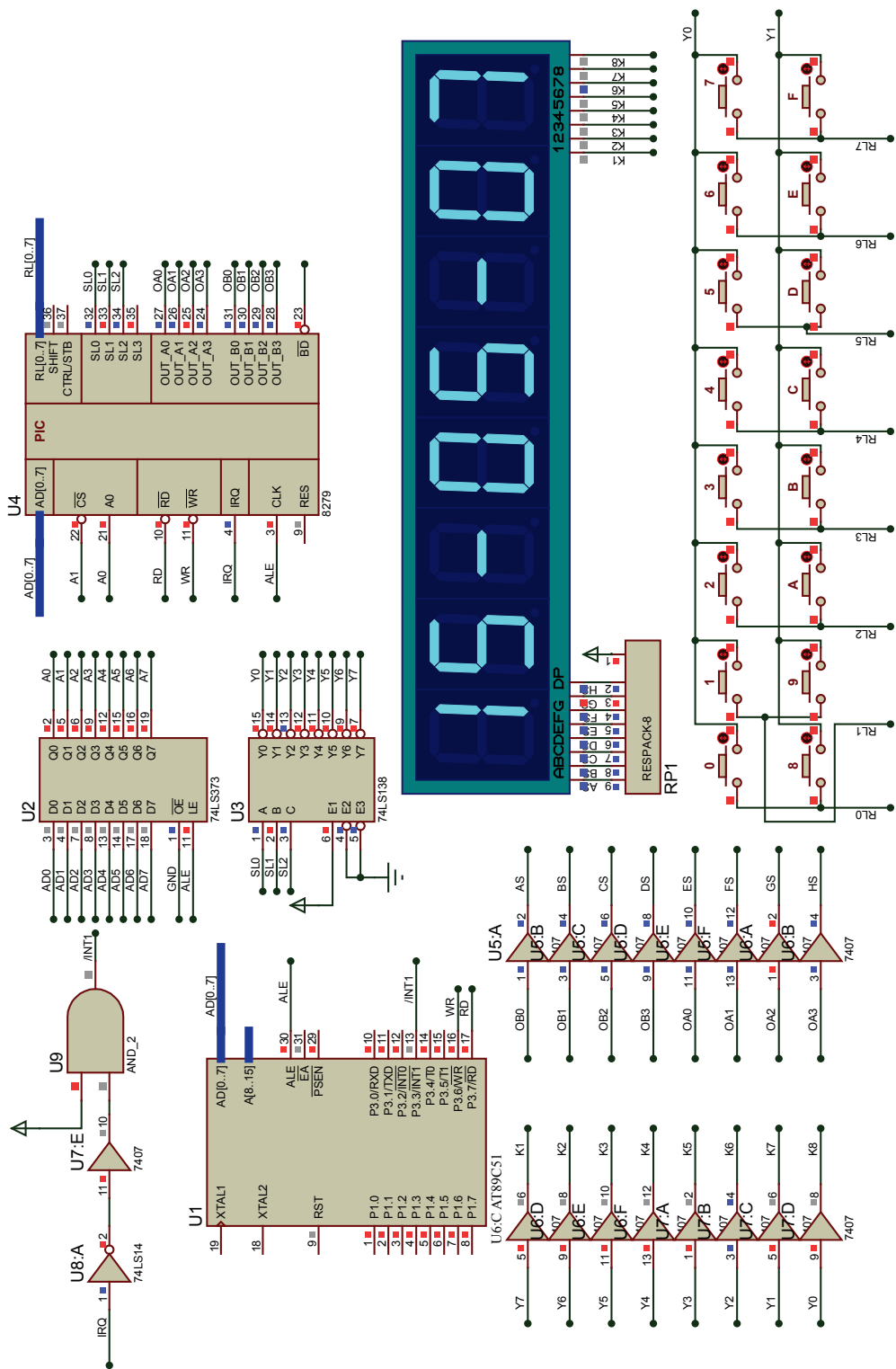


图 10.3 基于单片机的电子日历设计仿真电路

四、程序设计

```

//-----头文件引用-----
#include<reg51.h>
#include<absacc.h>
#include<intrins.h>
//-----宏声明-----
#define D8279    XBYTE[0xFFFC]    //8279 数据口地址
#define C8279    XBYTE[0xFFFD]    //8279 状态/命令口地址
#define uchar    unsigned char
#define uint     unsigned int
#define TimeDisp 1
#define DateDisp 0
//-----变量定义-----
uchar idata time[] = {0,0,0,12}; //10 毫秒, 秒, 分, 十
uchar idata day[3] = {18,6,8};
uchar idata diss[8]={0x20,0x20,0,0,0,0,0,0}; //显示缓冲区
uchar                                     code                                     ledseg[] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5E,0x79
,0x71,0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,0x87,0xFF,0xEF,0xF7,0xFC,0xB9,0xDE
,0xF9,0xF1,0x00,0x40}; //LED 显示常数表
bit bdata sbz,wbz,kbz; //定义秒标志、键值合法标志、读键标志
bit bdata flag = TimeDisp;
//-----取键值函数-----
uchar getkey()
{
    uchar key;
    if((C8279&0x07)==0x00)
        {kbz=0;} //无键按下, 置标志
    else
    {
        kbz=1;
        C8279=0x40; //读 FIFO RAM 命令
        key=D8279;
        key=(key&0x3f); //取键盘数据低 6 位
    }
    return(key);
}
//-----显示函数-----
void disled(d)
    uchar idata *d;
{
    uchar i;C8279=0x90;
    for(i=0;i<8;i++)
    {
        C8279=i+0x80;
        D8279=ledseg[*d];
        d++;
    }
}

```

```
    }
}
//-----显示缓冲区内容显示-----
void disp(void)
{
    disled(diss);
}
//-----8279 初始化子程序-----
void init8279()
{
    C8279=0;          //置 8279 工作方式
    C8279=0x2f;        //置键盘扫描速率
    C8279=0xc1;        //清除 LED 显示
    //while (com&0x80); //等待清除结束
}
//-----毫秒显示-----
void disms ()
{
    diss[0]=time[0]%10;
    diss[1]=time[0]/10;
    disp();
}
//-----显示处理-----
void display()
{
    if(flag == 1)
    {
        diss[0]=time[1]%10;
        diss[1]=time[1]/10;
        diss[2]=33;
        diss[3]=time[2]%10;
        diss[4]=time[2]/10;
        diss[5]=33;
        diss[6]=time[3]%10;
        diss[7]=time[3]/10;
    }
    else
    {
        diss[0]=day[0]%10;
        diss[1]=day[0]/10;
        diss[2]=33;
        diss[3]=day[1]%10;
        diss[4]=day[1]/10;
        diss[5]=33;
        diss[6]=day[2]%10;
        diss[7]=day[2]/10;
    }
    disp();
}
```

```

//-----初始化-----
void first(void)
{
    init8279();    //初始化 8279
    sbz=1;        //标志

    TMOD=0x10;
    TH1=0xdc;     //10 毫秒的时间常数
    TL1=0x00;
    disms();
    ET1=1;EA=1;   //允许中断
}
//-----INT_T1 中断服务子程序-----
void Int_T1(void) interrupt 3
{
    TR1=0;
    TH1=0xdc;     //10 毫秒定时常数
    TL1=0x00;
    TR1=1;
    time[0]=time[0]+1;    //10 毫秒数加 1
    if(time[0]==100)      //判断 10 毫秒=100
    {
        time[0]=0;
        sbz=1;          //置秒标志
    }
    //disms();
    if(time[0]==0)
    {
        time[1]=time[1]+1; //秒加 1
        if(time[1]==60)    //判断秒=60
        {
            time[1]=0;
            time[2]=time[2]+1; //分加 1
            if(time[2]==60)    //判断分=60
            {
                time[2]=0;
                time[3]=time[3]+1; //时加 1
                if(time[3]==24) //判断时=23
                {
                    time[3]=0;
                    day[0] += 1;

                    if(day[0] == 30)
                    {
                        day[0] = 0;
                        day[1] += 1;
                        if(day[1] == 12)

```



```

        day[1] = 0;
        day[2]++;
    }
}
}
}
}
}
}
//-----读数子程序-----
void getword()
{
    uchar i;
    for(i=8;i>0;i--)
    {
        do
        {
            getkey();    //读键盘
        }
        while(kbz==0); //无键输入, 则再读

        if((getkey()>9)|| (getkey()<0)) //判断输入是否大于 9, 小于 0
        {
            wbz=0;    //置非法输入标志
            return;
        }
        else
        {
            wbz=1;    //置合法输入标志
            if((i == 6)|| (i == 3))
            {

                i--;
                diss[i-1]=getkey();
                disp();    //显示输入的字符
            }
            else
            {

                diss[i-1]=getkey();
                disp();    //显示输入的字符
            }
        }
    }
}
//-----时间清零子程序-----
void cleart()
{

```

```

    TR1=0;        //关计数器
    time[0]=0;    //10 毫秒清零
    time[1]=0;    //秒清零
    time[2]=0;    //分钟清零
    time[3]=0;    //小时清零
    disms();      //显示毫秒
    sbz=1;        //置秒标志
}
//-----设置初值子程序-----
void sett()
{
    getword();    //读数
    if(wbz==1)    //判断输入合法性
    {
        if(flag == 1)
        {
            time[3]=(diss[7]*10+diss[6]);
            if(time[3]<24)//判断输入小时值 < 24
            {
                time[2]=(diss[4]*10+diss[3]);
                if(time[2]<60)//判断输入分钟数 < 60

                {
                    time[1]=(diss[1]*10+diss[0]);
                    if(time[1]<60)//判断输入秒值 < 60
                    {

                        ;
                    }
                }
                else
                {
                    cleart();
                } //时间清零
            }
        }
        else
        {
            cleart();
        } //时间清零
    }
    else
    {
        cleart();
    } //时间清零
}
    else
    {
        day[2]=(diss[7]*10+diss[6]);
        if(day[2]<32)//判断输入小时值 < 24
        {

```

```

        day[1]=(diss[4]*10+diss[3]);
        if(day[1]<13)//判断输入分钟数 < 60
        {
            day[0]=(diss[1]*10+diss[0]);

            if(day[0]<100)//判断输入秒值 < 60
            {
                ;
            }

            else
            {
                cleart();
            } //时间清零
        }
        else
        {
            cleart();
        } //时间清零
    }
    else
    {
        cleart();
    } //时间清零
}

//-----主程序-----
void main()
{
    uint counter = 0;first();    //初始化
    while(1)    //循环
    {

        counter++;
        if(counter == 10000 )
        {    //flag = !flag;
            counter = 0;
        }

        getkey();    //读键盘
        if(kbz==1)    //判断是否有键输入
        {
            switch(getkey())

```

```
{
    case 0x0c: cleart(); //输入键是'C',转 break;
    case 0x0a: TR1=1; //输入键是'A',电子钟计时
    break;
    case 0x0d: TR1=0; //输入键是'D',电子钟停止计时

    break;
    case 0x0b: TR1=0; //输入键是'B',转 SETT
    sett();
    break;
    case 0x0e: _nop_(); //输入键是'E',
    case 0x0f: flag = !flag;
    break;
    while(1) //等待回到监控
        { }
    }
else
    if(sbz==1)
        { display(); //显示时间
          sbz=0; //清标志
        }
    }
}
```

五、实例小结

通过本课程设计,掌握以 MCS-51 单片机为核心的电子电路的设计方法和应用技术,掌握 8279 键盘显示电路的编程方法,进一步掌握定时器的使用和编程方法及中断处理程序的编程方法。

实例四 基于单片机的具备温度显示的数字时钟设计

一、设计要求

基于 DS1302 和数字温度传感器 DS18B20 设计一个带温度显示的万年历。

二、实例分析

采用单片机 AT89C51 与集成温度传感器 DS18B20、时钟芯片 DS1302、液晶显示器 LCD1602 构成数字时钟和温度计。通过编写程序实现对 DS18B20、DS1302 的读写操作,实现时间、温度等数据在液晶显示器上的正确显示。

三、硬件仿真电路

基于单片机的具备温度显示的数字时钟设计仿真电路如图 10.4 所示。

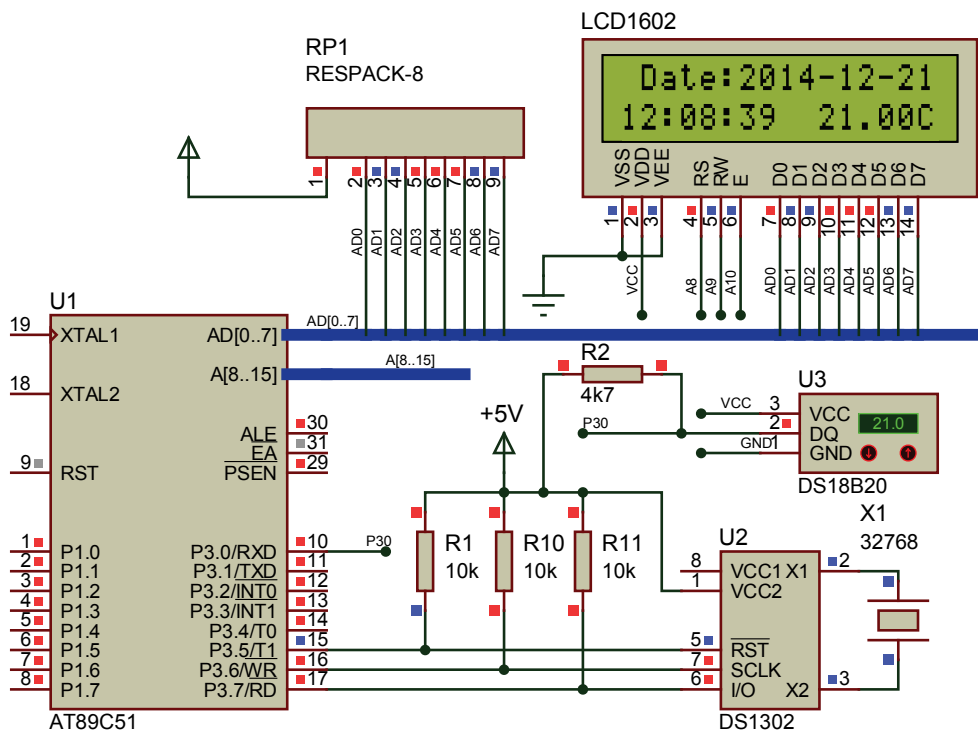


图 10.4 基于单片机的具备温度显示的数字时钟设计仿真电路

四、程序设计

```
#include<reg51.h>    //包含单片机寄存器的头文件
#include<intrins.h>  //包含_nop_()函数定义的头文件
#include<stdio.h>
/*****
以下是 DS1302 芯片的操作程序
*****/
unsigned char code dat[10]={"0123456789"};
sbit RST=P3^5;
sbit SCLK=P3^6;
sbit DATA=P3^7;
bit ReadTempFlag; //定义读时间标志
unsigned int ReadTemperature(void);
/*****
函数功能：延时若干微秒
*****/
void delaynus(unsigned char n)
```

```

{
    unsigned char i;
    for(i=0;i<n;i++)
        ;
}
/*****
函数功能：向 1302 写一个字节数据
*****/
void Write1302(unsigned char dat)
{
    unsigned char i;
    SCLK=0;
    delaynus(2);
    for(i=0;i<8;i++)
    {
        DATA=dat&0x01;
        delaynus(2);
        SCLK=1;
        delaynus(2);
        SCLK=0;
        dat>>=1;
    }
}

/*****
函数功能：根据命令字，向 1302 写一个字节数据
*****/
void WriteSet1302(unsigned char Cmd,unsigned char dat)
{
    RST=0;
    SCLK=0;
    RST=1;
    delaynus(2);
    Write1302(Cmd);
    Write1302(dat);
    SCLK=1;
    RST=0;
}

/*****
函数功能：从 1302 读一个字节数据
*****/
unsigned char Read1302(void)
{
    unsigned char i,dat;
    delaynus(2);
    for(i=0;i<8;i++)
    {
        dat>>=1;
        if(DATA==1)
            dat|=0x80;
        SCLK=1;
    }
}

```

```

        delaynus(2);
        SCLK=0;
        delaynus(2);
    }
    return dat;
}
/*****
函数功能: 根据命令字, 从 1302 读取一个字节数据
*****/
unsigned char ReadSet1302(unsigned char Cmd)
{
    unsigned char dat;
    RST=0;
    SCLK=0;
    RST=1;
    Write1302(Cmd);
    dat=Read1302();
    SCLK=1;
    RST=0;
    return dat;
}
/*****
函数功能: 1302 进行初始化设置
*****/
void Init_DS1302(void)
{
    WriteSet1302(0x8E,0x00);
    WriteSet1302(0x80,((0/10)<<4|(0%10)));
    WriteSet1302(0x82,((0/10)<<4|(8%10)));
    WriteSet1302(0x84,((12/10)<<4|(2%10)));
    WriteSet1302(0x86,((20/10)<<4|(1%10)));
    WriteSet1302(0x88,((12/10)<<4|(2%10)));
    WriteSet1302(0x8c,((10/10)<<4|(4%10)));
}
/*****
以下是对液晶模块的操作程序
*****/
sbit RS=P2^0;
sbit RW=P2^1;
sbit E=P2^2;
sbit BF=P0^7;
/*****
函数功能: 延时 1ms
(3j+2)*i=(3×33+2)×10=1010(微秒), 可以认为是 1 毫秒
*****/
void delay1ms()
{
    unsigned char i,j;
    for(i=0;i<10;i++)
        for(j=0;j<33;j++)
            ;
}

```

```

}
/*****
函数功能：延时若干毫秒
*****/
void delaynms(unsigned char n)
{
    unsigned char i;
    for(i=0;i<n;i++)
        delaylms();
}
/*****
函数功能：判断液晶模块的忙碌状态
*****/
bit BusyTest(void)
{
    bit result;
    RS=0;
    RW=1;
    E=1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    result=BF;
    E=0;
    return result;
}
/*****
函数功能：将模式设置指令或显示地址写入液晶模块
*****/
void WriteInstruction (unsigned char dictate)
{
    while(BusyTest()==1);
    RS=0;
    RW=0;
    E=0;

    _nop_();
    _nop_();
    P0=dictate;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    E=1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    E=0;
}

```



```

/*****
函数功能：指定字符显示的实际地址
*****/
void WriteAddress(unsigned char x)
{
    WriteInstruction(x|0x80);
}
/*****
函数功能：将数据(字符的标准 ASCII 码)写入液晶模块
*****/
void WriteData(unsigned char y)
{
    while(BusyTest()==1);
    RS=1;
    RW=0;
    E=0;

    P0=y;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    E=1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    E=0;
}
/*****
函数功能：对 LCD 的显示模式进行初始化设置
*****/
void LcdInitiate(void)
{
    delaynms(15);
    WriteInstruction(0x38);
    delaynms(5);
    WriteInstruction(0x38);
    delaynms(5);
    WriteInstruction(0x38);
    delaynms(5);
    WriteInstruction(0x0c);
    delaynms(5);
    WriteInstruction(0x06);
    delaynms(5);
    WriteInstruction(0x01);
    delaynms(5);
}

```

```

}
/*****
以下是 1302 数据的显示程序
*****/
/*****
函数功能：显示秒
*****/
void DisplaySecond(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位
    j=x%10; //取个位
    WriteAddress(0x46);     //写显示地址,将在第 2 行第 1 列开始显示
    WriteData(dat[i]);      //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);      //将十位数字的字符常量写入 LCD
    delaynms(50);          //延时 1ms 给硬件一点反应时间
}

/*****
函数功能：显示分钟
入口参数：x
*****/
void DisplayMinute(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位
    j=x%10; //取个位
    WriteAddress(0x43);     //写显示地址,将在第 2 行第 7 列开始显示
    WriteData(dat[i]);      //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);      //将十位数字的字符常量写入 LCD
    delaynms(50);          //延时 1ms 给硬件一点反应时间
}

/*****
函数功能：显示小时
入口参数：x
*****/
void DisplayHour(unsigned char x)
{
    unsigned char i,j;      //j,k,l 分别储存温度的百位、十位和个位
    i=x/10; //取十位
    j=x%10; //取个位
    WriteAddress(0x40);     //写显示地址,将在第 2 行第 7 列开始显示
    WriteData(dat[i]);      //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);      //将十位数字的字符常量写入 LCD
    delaynms(50);          //延时 1ms 给硬件一点反应时间
}

/*****
函数功能：显示日
入口参数：x
*****/

```

```

void DisplayDay(unsigned char x)
{
    unsigned char i,j;    //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x0e);    //写显示地址,将在第 2 行第 7 列开始显示
    WriteData(dat[i]);    //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);    //将十位数字的字符常量写入 LCD
    delaynms(50);    //延时 1ms 给硬件一点反应时间
}
/*****

```

函数功能: 显示月

入口参数: x

*****/

```

void DisplayMonth(unsigned char x)
{
    unsigned char i,j;    //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x0b);    //写显示地址,将在第 2 行第 7 列开始显示
    WriteData(dat[i]);    //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);    //将十位数字的字符常量写入 LCD
    delaynms(50);    //延时 1ms 给硬件一点反应时间
}
/*****

```

函数功能: 显示年

入口参数: x

*****/

```

void DisplayYear(unsigned char x)
{
    unsigned char i,j;    //j,k,l 分别储存温度的百位、十位和个位
    i=x/10;//取十位
    j=x%10;//取个位
    WriteAddress(0x08);    //写显示地址,将在第 2 行第 7 列开始显示
    WriteData(dat[i]);    //将百位数字的字符常量写入 LCD
    WriteData(dat[j]);    //将十位数字的字符常量写入 LCD
    delaynms(50);    //延时 1ms 给硬件一点反应时间
}

```

*****/

函数功能: 主函数

*****/

```

void main(void)
{
    int temp;
    float temperature;
    char displaytemp[16]; //定义显示区域临时存储数组
    unsigned char second,minute,hour,day,month,year;    //分别储存秒、分、小
    时、日、月、年
    unsigned char ReadValue;    //储存从 1302 读取的数据

```

```

LcdInitiate();           //将液晶初始化
WriteAddress(0x01);      //写 Date 的显示地址,将在第 1 行第 2 列开始显示
WriteData('D');          //将字符常量写入 LCD
WriteData('a');          //将字符常量写入 LCD
WriteData('t');          //将字符常量写入 LCD
WriteData('e');          //将字符常量写入 LCD
WriteData(':');           //将字符常量写入 LCD

WriteAddress(0x06);      //写年月分隔符的显示地址,显示在第 1 行第 9 列
WriteData('2');          //将字符常量写入 LCD
WriteAddress(0x07);      //写年月分隔符的显示地址,显示在第 1 行第 9 列
WriteData('0');          //将字符常量写入 LCD

WriteAddress(0x0a);      //写年月分隔符的显示地址,显示在第 1 行第 9 列
WriteData('-');          //将字符常量写入 LCD
WriteAddress(0x0d);      //写月日分隔符的显示地址,显示在第 1 行第 12 列
WriteData('-');          //将字符常量写入 LCD
WriteAddress(0x42);      //写小时与分钟分隔符的显示地址,显示在第 2 行第 6 列
WriteData(':');          //将字符常量写入 LCD
WriteAddress(0x45);      //写分钟与秒分隔符的显示地址,显示在第 2 行第 9 列
WriteData(':');          //将字符常量写入 LCD
Init_DS1302();           //将 1302 初始化
while(1)
{
    ReadValue = ReadSet1302(0x81);    //从秒寄存器读数据
    second=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplaySecond(second);            //显示秒
    ReadValue = ReadSet1302(0x83);    //从分寄存器读
    minute=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplayMinute(minute);            //显示分
    ReadValue = ReadSet1302(0x85);    //从分寄存器读
    hour=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplayHour(hour);                //显示小时
    ReadValue = ReadSet1302(0x87);    //从分寄存器读
    day=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplayDay(day);                  //显示日
    ReadValue = ReadSet1302(0x89);    //从分寄存器读
    month=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplayMonth(month);              //显示月
    ReadValue = ReadSet1302(0x8d);    //从分寄存器读
    year=((ReadValue&0x70)>>4)*10 + (ReadValue&0x0F); //将读出数据转化
    DisplayYear(year);                //显示年

    //if(ReadTempFlag==1)
    //{
    ReadTempFlag=0;
    temp=ReadTemperature();
    temperature=(float)temp*0.0625;
    sprintf(displaytemp,"Temp % 7.3f",temperature); //打印温度值

```

```
WriteAddress(0x49);  
WriteData(displaytemp[6]);      //将字符常量写入 LCD  
WriteAddress(0x4a);  
WriteData(displaytemp[7]);      //将字符常量写入 LCD  
WriteAddress(0x4b);  
WriteData(displaytemp[8]);      //将字符常量写入 LCD  
WriteAddress(0x4c);  
WriteData(displaytemp[9]);      //将字符常量写入 LCD  
WriteAddress(0x4d);  
WriteData(displaytemp[10]);     //将字符常量写入 LCD  
WriteAddress(0x4e);  
WriteData(displaytemp[11]);     //将字符常量写入 LCD  
WriteAddress(0x4f);  
WriteData('C');                 //将字符常量写入 LCD  
//}  
}  
}
```

五、实例小结

采用 AT89C51 作为主控制系统；DS1302 提供时钟；数字式温度传感器；LED 数码管动态扫描作为显示。在硬件电路方面，采用专业的时钟芯片 DS1302，它可以对年、月、日、周日、时、分、秒进行计时，具有闰年补偿功能。主要特点是采用串行数据传输，可为掉电保护电源提供可编程的充电功能；在显示模块的选择上，选择了 LCD1602，显示清晰，测量温度采用 DS18B20。

实例五 基于单片机的具备转速显示功能的直流电动机控制系统设计

一、设计要求

直流电动机控制系统的主要功能包括：直流电动机的加速、减速及正转、反转和停止，可以调整电动机的转速，可以方便地读出电动机转速的大小，能够很方便地实现电动机的智能控制。请查阅 L298N 和 LCD 相关资料完成设计。

二、实例分析

用 AT89C51 单片机为核心，通过程序控制由单片机产生不同的 PWM(脉冲宽度调制)信号，送给电动机驱动芯片 L298N 的使能端口，通过 L298 驱动芯片来控制直流电动机的启动、速度、方向的变化；单片机将速度传送给 LM016L 显示。整个电路设计包括单片机控制模块、速度显示模块和电动机及电动机驱动模块。

三、硬件仿真电路

基于单片机的具备转速显示功能的直流电动机控制系统设计仿真电路如图 10.5 所示。

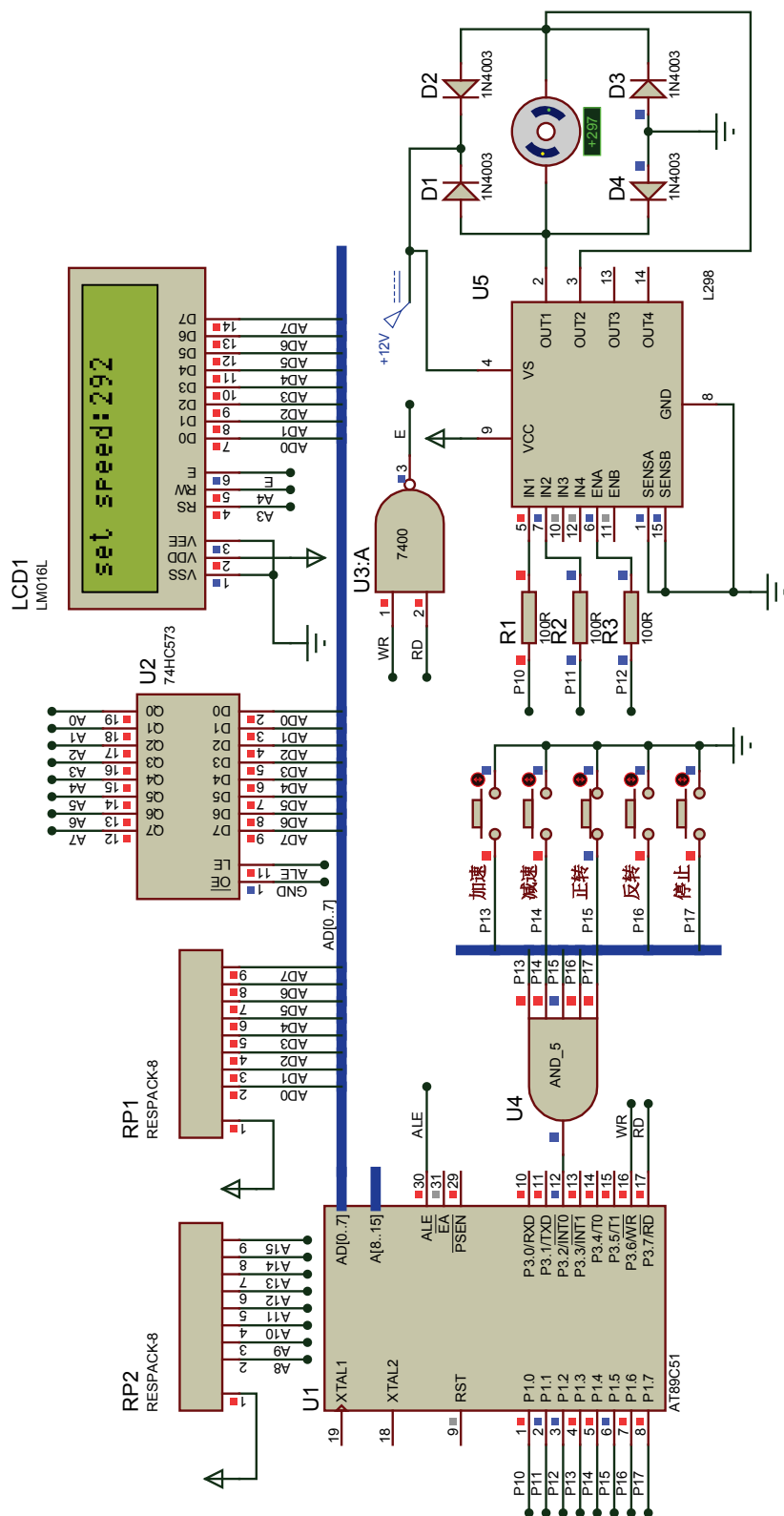


图 10.5 基于单片机的具备转速显示功能的直流电动机控制系统设计仿真电路

四、程序设计

```
#include<reg51.h>
#include<absacc.h>
#define CMD_WR XBYTE[0x00]
#define DATA_WR XBYTE[0x08]
#define BUSY_RD XBYTE[0x10]
#define DATA_RD XBYTE[0x18]
#define CLS 0x01
#define HOME 0x02
#define SETMODE 0x06
#define SETVISIBLE 0x08
#define SHIFT 0x10
#define SETFUNCTION 0x38
#define SETCGADDR 0x40
#define SETDDADDR 0x80
char lcd[]={"set speed:292"};
void busy(void);
void Delay(unsigned char j);
void lcdini(void)
{
    Delay(20);
    busy();
    CMD_WR=SETFUNCTION;
    busy();
    CMD_WR=SETVISIBLE;
    busy();
    CMD_WR=CLS;
    busy();
    CMD_WR=SETMODE;
    busy();
    CMD_WR=0x0c;
}

void busy(void)
{
    Delay(5);
}

void Delay(unsigned char j)
{
    unsigned char i;
    while(--j!=0)
    {
        for(i=0; i<125; i++); //耗时 1 毫秒
    }
}
```

```
#include<lcd.h>
sbit IN1=P1^0;
sbit IN2=P1^1;
sbit ENA=P1^2;
sbit Q=P1^3;
sbit S=P1^4;
sbit Z=P1^5;
sbit F=P1^6;
sbit STO=P1^7;
void pwm(unsigned char t)
{
    ENA=0;
    Delay(200);
    ENA=1;
    Delay(t);
}

void zz()
{
    IN1=1;
    IN2=0;
}
void fz()
{
    IN1=0;
    IN2=1;
}
void stop()
{
    IN1=0;
    IN2=0;
}

void main(void)
{
    unsigned char i;
    lcdini();
    EA=1;
    EX0=1;
    IT0=1;
    PX0=1;
    P1=0xff;
    while(1)
    {
        busy();
        CMD_WR=0x80;
        for(i=0;i<14;i++)
        {
            busy();
            DATA_WR=lcd[i];
```



```

    }
}

void int00() interrupt 0 using 0
{
    while(1)
    {
        switch(P1)
        {
            case 0xf7:zz();pwm(255);break;
            case 0xef:zz();pwm(10);break;
            case 0xdf:zz();pwm(200);break;
            case 0xbf:fz();pwm(200);break;
            case 0x7f:stop();break;
            default:P1=0xff;break;
        }
    }
}

```

五、实例小结

可以利用红外传感器测直流电动机的转速，控制直流电动机的转动速度，用 PWM 调速方式控制直流电动机转动的速度，以及正反转动，并可以自动调节速度至预先设定的速度。整个系统的电路逻辑结构简单，可靠性能高，实现功能强。

实例六 基于单片机的红外遥控器控制继电器的设计

一、设计要求

要求用遥控器控制实验板上的继电器。

二、实例分析

通用红外遥控系统由发射和接收两大部分组成。应用编/解码专用集成电路芯片来进行控制操作。发射部分包括键盘矩阵、编码调制、LED 红外发送器；接收部分包括光、电转换放大器、解调、解码电路。

红外遥控器遥控信号波形：for PanasonicTV，如图 10.6 所示。

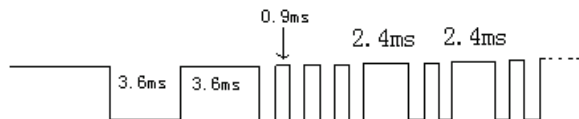


图 10.6 红外遥控器波形图

三、硬件接线图

基于单片机的红外遥控器控制继电器的设计实物连接图如图 10.7 所示。

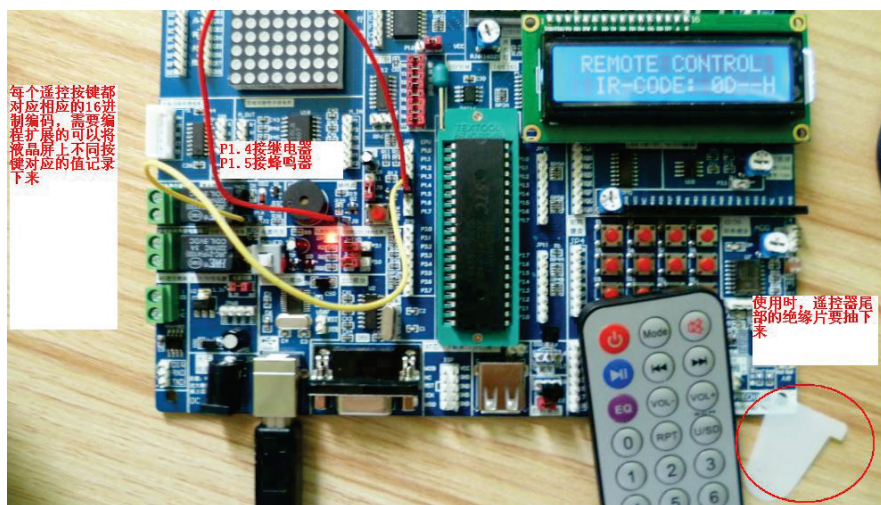


图 10.7 基于单片机的红外遥控器控制继电器的设计实物连接图

四、程序设计

```

/*****
***
* 描述:
*      lcd1602 显示 遥控键值读取器
*      lcd1602 显示 遥控器接 p3.2
*      喇叭接 p1.5 继电器接 p1.4
*      连接方法: 使用红外功能时 J1 跳线短接
*
*****
**/

#include <reg51.h>
#include <intrins.h>
#define NOP() _nop_() /* 定义空指令 */
#define uchar unsigned char
#define uint unsigned int
#define delayNOP(); {_nop_();_nop_();_nop_();_nop_();};
void delay(uchar x); //x*0.14MS
void delay1(int ms);
void beep(void);
sbit IRIN = P3^2; //红外接收器数据线
sbit RELAY= P1^4; //继电器驱动线
sbit BEEP = P1^5; //蜂鸣器驱动线
uchar IRCOM[7];
uchar cdis1[] = {" REMOTE CONTROL "};
uchar cdis2[] = {" IR-CODE: ----H"};

```

```

//LCD IO
sbit LCD_RW = P2^5;
sbit LCD_RS = P2^6;
sbit LCD_EN = P2^7;

unsigned char Y0;

/*****
/*
/*检查 LCD 忙状态
/*lcd_busy 为 1 时, 忙, 等待。lcd_busy 为 0 时, 闲, 可写指令与数据。
/*
/*
*****/

bit lcd_busy()
{
    bit result;
    LCD_RS = 0;
    LCD_RW = 1;
    LCD_EN = 1;
    delayNOP();
    result = (bit) (P0&0x80);
    LCD_EN = 0;
    return(result);
}
/*****
/*
/*写指令数据到 LCD
/*RS=L, RW=L, E=高脉冲, D0-D7=指令码。
/*
/*
*****/

void lcd_wcmd(uchar cmd)
{
    while(lcd_busy());
    LCD_RS = 0;
    LCD_RW = 0;
    LCD_EN = 0;
    _nop_();
    _nop_();
    P0 = cmd;
    delayNOP();
    LCD_EN = 1;
    delayNOP();
    LCD_EN = 0;
}
/*****
/*
/*写显示数据到 LCD
/*RS=H, RW=L, E=高脉冲, D0-D7=数据。
/*
/*
*****/

```

```

/*****/
void lcd_wdat(uchar dat)
{
    while(lcd_busy());
    LCD_RS = 1;
    LCD_RW = 0;
    LCD_EN = 0;
    P0 = dat;
    delayNOP();
    LCD_EN = 1;
    delayNOP();
    LCD_EN = 0;
}
/*****/
/*                                     */
/* LCD 初始化设定                      */
/*                                     */
/*****/
void lcd_init()
{
    delay1(15);
    lcd_wcmd(0x38);      //16*2 显示, 5*7 点阵, 8 位数据
    delay1(5);
    lcd_wcmd(0x38);
    delay1(5);
    lcd_wcmd(0x38);
    delay1(5);
    lcd_wcmd(0x0c);      //显示开, 关光标
    delay1(5);
    lcd_wcmd(0x06);      //移动光标
    delay1(5);
    lcd_wcmd(0x01);      //清除 LCD 的显示内容
    delay1(5);
}
/*****/
/*                                     */
/* 设定显示位置                        */
/*                                     */
/*****/
void lcd_pos(uchar pos)
{
    lcd_wcmd(pos | 0x80); //数据指针=80+地址变量
}

/*****/
main()
{
    uchar m;
    IE = 0x81;           //允许总中断中断, 使能 INT0 外部中断
    TCON = 0x01;         //触发方式为脉冲负边沿触发
}

```

```

    IRIN=1;                //I/O 口初始化
    BEEP=1;
    RELAY=1;
    delay1(10);            //延时
    lcd_init();            //初始化 LCD
    lcd_pos(0);            //设置显示位置为第一行的第 1 个字符
    m = 0;
    while(cdis1[m] != '\0')
    {                        //显示字符
        lcd_wdat(cdis1[m]);
        m++;
    }
    lcd_pos(0x40);         //设置显示位置为第二行第 1 个字符
    m = 0;
    while(cdis2[m] != '\0')
    {                        //显示字符
        lcd_wdat(cdis2[m]);
        m++;
    }
    while(1);
} //end main
/*****
void IR_IN(void) interrupt 0    //外部中断服务程序
{
    unsigned char j,k,N=0;
        EX0 = 0;
        delay(15);
        if (IRIN==1)
        { EX0 =1;
          return;
        }

        //确认 IR 信号出现
        while (!IRIN)        //等 IR 变为高电平, 跳过 9ms 的前导低电平信号。
        {delay(1);}
        for (j=0;j<4;j++)    //收集四组数据
        {
            for (k=0;k<8;k++) //每组数据有 8 位
            {
                while (IRIN) //等 IR 变为低电平, 跳过 4.5ms 的前导高电平信号。
                {delay(1);}
                while (!IRIN) //等 IR 变为高电平
                {delay(1);}
                while (IRIN) //计算 IR 高电平时长
                {
                    delay(1);
                    N++;
                    if (N>=30)
                    { EX0=1;
                      return;} //0.14ms 计数过长自动离开。
                }
            }
        }
    }
    //高电平计数完毕

```

```


    IRCOM[j]=IRCOM[j] >> 1;           //数据最高位补“0”
    if (N>=8) {IRCOM[j] = IRCOM[j] | 0x80;} //数据最高位补“1”
    N=0;
} //end for k
} //end for j
if (IRCOM[2] != ~IRCOM[3])
{ EX0=1;
  return; }
IRCOM[5]=IRCOM[2] & 0x0F;           //取键码的低四位
IRCOM[6]=IRCOM[2] >> 4;           //右移 4 次，高四位变为低四位
if (IRCOM[5]>9)
{ IRCOM[5]=IRCOM[5]+0x37; }
else
  IRCOM[5]=IRCOM[5]+0x30;
if (IRCOM[6]>9)
{ IRCOM[6]=IRCOM[6]+0x37; }
else
  IRCOM[6]=IRCOM[6]+0x30;
lcd_pos(0x4b);
lcd_wdat(IRCOM[6]);           //第一位数显示
lcd_pos(0x4c);
lcd_wdat(IRCOM[5]);           //第二位数显示
Y0=0;
switch (IRCOM[2])
{
  case 0x09: Y0=0x01; break;
  case 0x1D: Y0=0x02; break;
  case 0x1F: Y0=0x03; break;
  case 0x0D: Y0=0x04; break;
  case 0x19: Y0=0x05; break;
  case 0x1B: Y0=0x06; break;
  case 0x11: Y0=0x07; break;
  case 0x15: Y0=0x08; break;
  case 0x17: Y0=0x09; break;
  case 0x13: RELAY=1; break;
  case 0x14: RELAY=1; break;
  case 0x51: RELAY=0; break;
}

if (Y0&0x01) RELAY=0; //打开继电器
else RELAY=1;         //关闭继电器
beep();
EX0 = 1;
}
/*****
void beep(void)
{
  unsigned char i;
  for (i=0; i<100; i++)
  {

```

```
    delay(4);
    BEEP=!BEEP;           //BEEP 取反
    }
    BEEP=1;               //关闭蜂鸣器
}
/*****
void delay(unsigned char x)    //x*0.14MS
{
    unsigned char i;
    while(x--)
    {
        for (i = 0; i<13; i++) {}
    }
}
*****/
void delay1(int ms)
{
    unsigned char y;
    while(ms--)
    {
        for(y = 0; y<250; y++)
        {
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
    }
}
```

五、实例小结

接收电路可以使用一种集红外线接收和放大于一体的一体化红外线接收器，不需要任何外接元件，就能完成从红外线接收到输出与 TTL 电平信号兼容的所有工作，而体积和普通的塑封三极管大小一样，它适合于各种红外线遥控和红外线数据传输。 

附录

附录 A MCS-51 系列单片机指令表

1. 数据传送指令

指令分类	助记符	功能说明	字节数	机器周期数
以 A 为目的操作数指令	MOV A, Rn	$A \leftarrow (Rn)$	1	1
	MOV A, direct	$A \leftarrow (direct)$	2	1
	MOV A, #data	$A \leftarrow data$	2	1
	MOV A, @Ri	$A \leftarrow (Ri)$	1	1
以 Rn 为目的操作数指令	MOV Rn, direct	$Rn \leftarrow (direct)$	2	2
	MOV Rn, #data	$Rn \leftarrow data$	2	1
	MOV Rn, A	$Rn \leftarrow (A)$	1	1
以直接地址为目的的操作数指令	MOV direct, Rn	$direct \leftarrow (Rn)$	2	2
	MOV direct, A	$direct \leftarrow (A)$	2	1
	MOV direct, @Ri	$direct \leftarrow (Ri)$	2	2
	MOV direct, #data	$direct \leftarrow data$	3	2
	MOV direct1, direct2	$direct1 \leftarrow (direct2)$	3	2
以寄存器间接地址为目的的操作数指令	MOV @Ri, A	$(Ri) \leftarrow (A)$	1	1
	MOV @Ri, direct	$(Ri) \leftarrow (direct)$	2	2
	MOV @Ri, #data	$(Ri) \leftarrow data$	2	1
16 位数据传送指令	MOV DPTR, #data16	$DPTR \leftarrow data16$	3	2
堆栈指令	PUSH direct	$sp \leftarrow (sp)+1, (sp) \leftarrow (direct)$	2	2
	POP direct	$direct \leftarrow ((sp)), sp \leftarrow (sp)-1$	2	2
数据交换指令	XCH A, Rn	$A \leftrightarrow Rn$	1	1
	XCH A, direct	$A \leftrightarrow (direct)$	2	1
	XCH A, @Ri	$A \leftrightarrow (Ri)$	1	1
	XCHD A, @Ri	$A \leftrightarrow (Ri)$	1	1
	SWAP A	$A_{7-4} \leftrightarrow A_{3-0}$	1	1
外部 RAM 数据传送指令	MOVX A, @DPTR	$A \leftarrow (DPTR)$	1	2
	MOVX @DPTR, A	$(DPTR) \leftarrow (A)$	1	2
	MOVX A, @Ri	$A \leftarrow (Ri)$	1	2
	MOVX @Ri, A	$(Ri) \leftarrow A$	1	2
查表指令	MOVC A, @A+DPTR	$A \leftarrow [(A)+(DPTR)]$	1	2
	MOVC A, @A+PC	$PC \leftarrow (PC)+1, A \leftarrow [(A)+(PC)]$	1	2

2. 算术运算指令

指令分类	助记符	功能说明	字节数	机器周期数
不带进位加法指令	ADD A, Rn	$A \leftarrow A + Rn$	1	1
	ADD A, direct	$A \leftarrow A + (\text{direct})$	2	1
	ADD A, #data	$A \leftarrow A + \text{data}$	2	1
	ADD A, @Ri	$A \leftarrow A + (Ri)$	1	1
带进位加法指令	ADDC A, Rn	$A \leftarrow A + Rn + Cy$	1	1
	ADDC A, direct	$A \leftarrow A + (\text{direct}) + Cy$	2	1
	ADDC A, #data	$A \leftarrow A + \text{data} + Cy$	2	1
	ADDC A, @Ri	$A \leftarrow A + (Ri) + Cy$	1	1
带借位减法指令	SUBB A, Rn	$A \leftarrow A - Rn - Cy$	1	1
	SUBB A, direct	$A \leftarrow A - \text{direct} - Cy$	2	1
	SUBB A, #data	$A \leftarrow A - \text{data} - Cy$	2	1
	SUBB A, @Ri	$A \leftarrow A - (Ri) - Cy$	1	1
加 1 指令	INC A	$A \leftarrow A + 1$	1	1
	INC Rn	$Rn \leftarrow Rn + 1$	1	1
	INC direct	$(\text{direct}) \leftarrow (\text{direct}) + 1$	2	1
	INC @Ri	$(Ri) \leftarrow (Ri) + 1$	1	1
	INC DPTR	$DPTR \leftarrow DPTR + 1$	1	2
减 1 指令	DEC A	$A \leftarrow A - 1$	1	1
	DEC Rn	$Rn \leftarrow Rn - 1$	1	1
	DEC direct	$(\text{direct}) \leftarrow (\text{direct}) - 1$	2	1
	DEC @Ri	$(Ri) \leftarrow (Ri) - 1$	1	1
乘法指令	MUL AB	$BA \leftarrow A \times B$	1	4
除法指令	DIV AB	$A(\text{商}), B(\text{余数}) \leftarrow A \div B$	1	4
十进制数调整指令	DA A	对 A 的内容进行 BCD 码调整	1	1

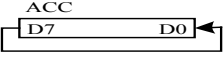
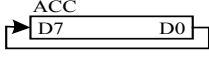
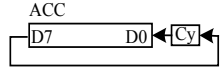
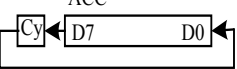
3. 逻辑运算指令

指令分类	助记符	功能说明	字节数	机器周期数
逻辑与运算指令	ANL A, Rn	$A \leftarrow A \wedge Rn$	1	1
	ANL A, direct	$A \leftarrow A \wedge (\text{direct})$	2	1
	ANL A, @Ri	$A \leftarrow A \wedge (Ri)$	1	1
	ANL A, #data	$A \leftarrow A \wedge \text{data}$	2	1
	ANL direct, A	$(\text{direct}) \leftarrow (\text{direct}) \wedge A$	2	1
	ANL direct, #data	$(\text{direct}) \leftarrow (\text{direct}) \wedge \text{data}$	3	2
逻辑或运算指令	ORL A, Rn	$A \leftarrow A \vee Rn$	1	1
	ORL A, direct	$A \leftarrow A \vee (\text{direct})$	2	1
	ORL A, @Ri	$A \leftarrow A \vee (Ri)$	1	1
	ORL A, #data	$A \leftarrow A \vee \text{data}$	2	1
	ORL direct, A	$(\text{direct}) \leftarrow (\text{direct}) \vee A$	2	1
	ORL direct, #data	$(\text{direct}) \leftarrow (\text{direct}) \vee \text{data}$	3	2

(续表)

指令分类	助记符	功能说明	字节数	机器周期数
逻辑异或运算指令	XRL A, Rn	$A \leftarrow A \oplus Rn$	1	1
	XRL A, direct	$A \leftarrow A \oplus (\text{direct})$	2	1
	XRL A, @Ri	$A \leftarrow A \oplus (Ri)$	1	1
	XRL A, #data	$A \leftarrow A \oplus \text{data}$	2	1
	XRL direct, A	$(\text{direct}) \leftarrow (\text{direct}) \oplus A$	2	1
	XRL direct, #data	$(\text{direct}) \leftarrow (\text{direct}) \oplus \text{data}$	3	2
清零	CLR A	$A \leftarrow 00$	1	1
取反	CPL A	$A \leftarrow \overline{A}$	1	1

4. 循环移位指令

指令分类	助记符	功能说明	字节数	机器周期数
不进位指令	RL A		1	1
	RR A		1	1
带进位指令	RRC A		1	1
	RLC A		1	1

5. 控制转移指令

指令分类	助记符	功能说明	字节数	机器周期数
条件转移指令	LJMP addr16	$PC \leftarrow \text{addr16}$	3	2
	AJMP addr16	PC_{15-11} 不变, $PC_{10-0} \leftarrow \text{addr}_{10-0}$	2	2
	SJMP rel	$PC \leftarrow PC + \text{rel}$	2	2
	JMP @A+DPTR	$PC \leftarrow A + DPTR$	1	2
条件转移指令	JZ rel	若 $A \neq 0$, 则 $PC \leftarrow PC + 2$; 若 $A = 0$, 则 $PC \leftarrow PC + 2 + \text{rel}$	2	2
	JNZ rel	若 $A = 0$, 则 $PC \leftarrow PC + 2$; 若 $A \neq 0$, 则 $PC \leftarrow PC + 2 + \text{rel}$	2	2
	JC rel	若 $Cy = 0$, 则 $PC \leftarrow PC + 2$; 若 $Cy = 1$, 则 $PC \leftarrow PC + 2 + \text{rel}$	2	2
	JNC rel	若 $Cy = 0$, 则 $PC \leftarrow PC + 2 + \text{rel}$; 若 $Cy = 1$, 则 $PC \leftarrow PC + 2$	2	2
	JB bit, rel	若 $(\text{bit}) = 0$, 则 $PC \leftarrow PC + 3$; 若 $(\text{bit}) = 1$, 则 $PC \leftarrow PC + 3 + \text{rel}$	3	2
	JNB bit, rel	若 $(\text{bit}) = 0$, 则 $PC \leftarrow PC + 3 + \text{rel}$; 若 $(\text{bit}) = 1$, 则 $PC \leftarrow PC + 3$	3	2
	JBC bit, rel	若 $(\text{bit}) = 0$, 则 $PC \leftarrow PC + 3$; 若 $(\text{bit}) = 1$, 则 $(\text{bit}) \leftarrow 0$ 后 $PC \leftarrow PC + 3 + \text{rel}$	3	2

(续表)

指令分类	助记符	功能说明	字节数	机器周期数
比较转移指令	CJNE A, #data, rel	若 A=data, 则 $PC \leftarrow PC+3$, $Cy \leftarrow 0$; 若 A > data, 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 0$; 若 A < data, 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 1$	3	2
	CJNE A, direct, rel	若 A=(direct), 则 $PC \leftarrow PC+3$, $Cy \leftarrow 0$; 若 A > (direct), 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 0$; 若 A < (direct), 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 1$	3	2
	CJNE Rn, #data, rel	若 Rn=data, 则 $PC \leftarrow PC+3$, $Cy \leftarrow 0$; 若 Rn > data 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 0$; 若 Rn < data 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 1$	3	2
	CJNE @Ri, #data, rel	若 (Ri)=data, 则 $PC \leftarrow PC+3$, $Cy \leftarrow 0$; 若 (Ri) > data, 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 0$; 若 (Ri) < data, 则 $PC \leftarrow PC+3+rel$, $Cy \leftarrow 1$	3	2
减 1 不等于 0 转移指令	DJNZ Rn, rel	① $Rn \leftarrow (Rn) - 1$ ② 若 $Rn \neq 0$, 则 $PC \leftarrow PC+2+rel$; 若 $(Rn) = 0$, 则 $PC \leftarrow PC+2$	2	2
	DJNZ direct, rel	① $direct \leftarrow (direct) - 1$ ② 若 $(direct) \neq 0$, 则 $PC \leftarrow PC+3+rel$; 若 $(direct) = 0$, 则 $PC \leftarrow PC+3$	3	2
调用指令	LCALL addr16	$PC \leftarrow PC+3$ $SP \leftarrow SP+1$ $SP \leftarrow PC_{7-0}$ $SP \leftarrow SP+1$ $SP \leftarrow PC_{15-8}$ $PC \leftarrow addr16$	3	2
	ACALL addr11	$PC \leftarrow PC+2$ $SP \leftarrow SP+1$ $SP \leftarrow PC_{7-0}$ $SP \leftarrow SP+1$ $SP \leftarrow PC_{15-8}$ $PC_{10-0} \leftarrow addr11$	2	2
子程序返回指令	RET	$PC_{15-8} \leftarrow (SP)$ $SP \leftarrow SP-1$ $PC_{7-0} \leftarrow SP$ $SP \leftarrow SP-1$	1	2
中断返回指令	RETI	$PC_{15-8} \leftarrow (SP)$ $SP \leftarrow SP-1$ $PC_{7-0} \leftarrow (SP)$ $SP \leftarrow SP-1$	1	2

6. 位指令

指令分类	助记符	功能说明	字节数	机器周期数
位传送指令	MOV C, bit	$Cy \leftarrow bit$	2	1
	MOV bit, C	$bit \leftarrow Cy$	2	2

(续表)

指令分类	助记符	功能说明	字节数	机器周期数
位清零指令	CLR C	$Cy \leftarrow 0$	1	1
	CLR bit	$bit \leftarrow 0$	2	1
位置 1 指令	SETB C	$Cy \leftarrow 1$	1	1
	SETB bit	$bit \leftarrow 1$	2	1
位操作指令	ANL C, bit	$Cy \leftarrow Cy \wedge bit$	2	2
	ANL C, /bit	$Cy \leftarrow Cy \wedge \overline{bit}$	2	2
	ORL C, bit	$Cy \leftarrow Cy \vee bit$	2	2
	ORL C, /bit	$Cy \leftarrow Cy \vee \overline{bit}$	2	2
	CPL C	$Cy \leftarrow \overline{Cy}$	1	1
	CPL bit	$bit \leftarrow \overline{bit}$	2	1

7. 空操作指令

指令分类	助记符	功能说明	字节数	机器周期数
空操作指令	NOP	$PC \leftarrow PC + 1$	1	1



附录 B Protreus 的常用元器件

7407 驱动门
1N914 二极管
74Ls00 与非门
74LS04 非门
74LS08 与门
74LS390 TTL 双十进制计数器
7SEG 4 针 BCD-LED 输出从 0~9 对应于 4 根线的 BCD 码
7SEG 3-8 译码器电路
BCD-7SEG[size=+0]转换电路
ALTERNATOR 交流发电机
AMMETER-MILLI mA 安培计
AND 与门
BATTERY 电池/电池组
BUS 总线
CAP 电容器
CAPACITOR 电容器
CLOCK 时钟信号源
CRYSTAL 晶振
D-FLIPFLOP D 触发器
FUSE 熔断丝
GROUND 地
LAMP 灯
LED-RED 红色发光二极管
LM016L 2 行 16 列液晶可显示 2 行 16 列英文字符，有 8 位数据总线 D0~D7，
RS，R/W，EN 三个控制端口（共 14 线），工作电压为 5V。没背光，和常用的 1602B
功能和引脚一样（除了调背光的二个线脚）
LOGIC ANALYSER 逻辑分析器
LOGICPROBE 逻辑探针
LOGICPROBE[BIG]逻辑探针用来显示连接位置的逻辑状态
LOGICSTATE 逻辑状态用鼠标点击，可改变该方框连接位置的逻辑状态
LOGICTOGGLE 逻辑触发
MASTERSWITCH 按钮 手动闭合，立即自动打开
MOTOR 马达

OR 或门
POT-LIN 三引线可变电阻器
POWER 电源
RES 电阻器
RESISTOR 电阻器
SWITCH 按钮 手动按一下一个状态
SWITCH-SPDT 二选通一按钮
VOLTMETER 伏特计
VOLTMETER-MILLI mV 伏特计
VTERM 串行口终端
Electromechanical 电动机
Inductors 变压器
Laplace Primitives 拉普拉斯变换
Memory Ics
Microprocessor Ics
Miscellaneous 各种器件 AERIAL-天线;
ATAHDD; ATMEGA64; BATTERY; CELL;
CRYSTAL-晶振; FUSE; METER-仪表;
Modelling Primitives 各种仿真器件 是典型的基本元器模拟, 不表示具体型号, 只用于仿真, 没有 PCB
Optoelectronics 各种发光器件 发光二极管, LED, 液晶, 等等
PLDs & FPGAs
Resistors 各种电阻器
Simulator Primitives 常用的器件
Speakers & Sounders
Switches & Relays 开关, 继电器, 键盘
Switching Devices 晶闸管
Transistors 晶体管 (三极管, 场效应晶体管)
TTL 74 series
TTL 74ALS series
TTL 74AS series
TTL 74F series
TTL 74HC series
TTL 74HCT series
TTL 74LS series
TTL 74S series
Analog Ics 模拟电路集成芯片
Capacitors 电容器集合
CMOS 4000 series

Connectors 排座, 排插
Data Converters ADC, DAC
Debugging Tools 调试工具
ECL 10000 Series

AND 与门
ANTENNA 天线
BATTERY 直流电源
BELL 铃, 钟
BVC 同轴电缆接插件
BRIDEG 1 整流桥 (二极管)
BRIDEG 2 整流桥 (集成块)
BUFFER 缓冲器
BUZZER 蜂鸣器
CAP 电容器
CAPACITOR 电容器
CAPACITOR POL 有极性电容器
CAPVAR 可调电容器
CIRCUIT BREAKER 熔断丝
COAX 同轴电缆
CON 插口
CRYSTAL 晶体振荡器
DB 并行插口
DIODE 二极管
DIODE SCHOTTKY 稳压二极管
DIODE VARACTOR 变容二极管
DPY_3-SEG 3 段 LED
DPY_7-SEG 7 段 LED
DPY_7-SEG_DP 7 段 LED (带小数点)
ELECTRO 电解电容器
FUSE 熔断器
INDUCTOR 电感器
INDUCTOR IRON 带铁芯电感器
INDUCTOR3 可调电感器
JFET N N 沟道场效应晶体管
JFET P P 沟道场效应晶体管
LAMP 灯泡
LAMP NEDN 起辉器
LED 发光二极管

METER 仪表
MICROPHONE 麦克风
MOSFET MOS 管
MOTOR AC 交流电动机
MOTOR SERVO 伺服电动机
NAND 与非门
NOR 或非门
NOT 非门
NPN NPN 三极管
NPN-PHOTO 感光三极管
OPAMP 运算放大器
OR 或门
PHOTO 感光二极管
PNP 三极管
NPN DAR NPN 三极管
PNP DAR PNP 三极管
POT 滑线变阻器
PELAY-DPDT 双刀双掷继电器
RES1.2 电阻器
RES3.4 可变电阻器
RESISTOR BRIDGE 桥式电阻器
RESPACK 电阻器
SCR 晶闸管
PLUG 插头
PLUG AC FEMALE 三相交流插头
SOCKET 插座
SOURCE CURRENT 电流源
SOURCE VOLTAGE 电压源
SPEAKER 扬声器
SW 开关
SW-DPDY 双刀双掷开关
SW-SPST 单刀单掷开关
SW-PB 按钮
THERMISTOR 电热调节器
TRANS1 变压器
TRANS2 可调变压器
TRIAC 三端双向可控硅
TRIODE 三极真空管
VARISTOR 变阻器

ZENER 齐纳二极管

DPY_7-SEG_DP 数码管

SW-PB 开关

Device.lib 包括电阻器、电容器、二极管、三极管和 PCB 的连接器符号

ACTIVE.LIB 包括虚拟仪器和有源器件

DIODE.LIB 包括二极管和整流桥

DISPLAY.LIB 包括 LCD、LED

BIPOLAR.LIB 包括三极管

FET.LIB 包括场效应晶体管

ASIMMDLS.LIB 包括模拟元器件

VALVES.LIB 包括电子管

ANALOG.LIB 包括电源调节器、运放和数据采样 IC

CAPACITORS.LIB 包括电容器

COMS.LIB 包括 4000 系列

ECL.LIB 包括 ECL10000 系列

MICRO.LIB 包括通用微处理器

OPAMP.LIB 包括运算放大器

RESISTORS.LIB 包括电阻器

FAIRCHLD .LIB 包括 FAIRCHLD 半导体公司的分立器件

LINTEC.LIB 包括 LINTEC 公司的运算放大器

NATDAC.LIB 包括国家半导体公司的数字采样器件

NATOA.LIB 包括国家半导体公司的运算放大器

TECOOR.LIB 包括 TECOOR 公司的 SCR 和 TRIAC

TEXOAC.LIB 包括德州仪器公司的运算放大器和比较器

ZETEX.LIB 包括 ZETEX 公司的分立器件 

附录 C C51 常用库函数

C51 编译器提供了丰富的库函数,使用库函数可以大大简化用户的程序设计工作从而提高编程效率,基于 MCS-51 系列单片机本身的特点,某些库函数的参数和调用格式与 ANSI C 标准有所不同。每个库函数都在相应的头文件中给出了函数原型声明,用户如果需要使用库函数,必须在源程序的开始处采用预处理命令 `#include`,将有关的头文件包含进来。下面是 C51 中常见的库函数。

1. 寄存器库函数 REG×××.H

在 REG×××.H 的头文件中定义了 MCS-51 的所有特殊功能寄存器和相应的位,定义时都用大写字母。当在程序的头部把寄存器库函数 REG×××.H 包含后,在程序中就可以直接使用 MCS-51 中的特殊功能寄存器和相应的位。

2. 字符函数 CTYPE.H

函 数 名	函数原型	功 能
isalpha	extern bit isalpha(char c);	检查参数字符是否为英文字母,是则返回 1,否则返回 0
isalnum	extern bit isalnum(char c);	检查参数字符是否为英文字母或数字字符,是则返回 1,否则返回 0
isctrl	extern bit isctrl(char c);	检查参数字符是否在 0x00~0x1f 之间或等于 0x7f,是则返回 1,否则返回 0
isdigit	extern bit isdigit(char c);	检查参数字符是否为数字字符,是则返回 1,否则返回 0
isgraph	extern bit isgraph(char c);	检查参数字符是否为可打印字符,可打印字符的 ASCII 值为 0x21~0x7e,是则返回 1,否则返回 0
isprint	extern bit isprint(char c);	除了与 isgraph 相同之外,还接收空格符(0x20)
ispunct	extern bit ispunct(char c);	检查参数字符是否为标点、空格和格式字符,是则返回 1,否则返回 0。
islower	extern bit islower(char c);	检查参数字符是否为小写英文字母,是则返回 1,否则返回 0
isupper	extern bit isupper(char c);	检查参数字符是否为大写英文字母,是则返回 1,否则返回 0
isspace	extern bit isspace(char c);	检查参数字符是否为空格、制表符、回车、换行、垂直制表符和送纸之一,是则返回 1,否则返回 0
isxdigit	extern bit isxdigit(char c);	检查参数字符是否为十六进制数字字符,是则返回 1,否则返回 0
toint	extern char toint(char c);	将 ASCII 字符的 0~9、A~F 转换为十六进制数,返回值为 0~F
tolower	extern char tolower(char c);	将大写字母转换成小写字母,如果不是大写字母,则不作转换直接返回相应的内容
toupper	extern char toupper(char c);	将小写字母转换成大写字母,如果不是小写字母,则不作转换直接返回相应的内容

3. 一般输入/输出函数 STDIO.H

C51 库中包含的输入/输出函数 STDIO.H 是通过 MCS-51 的串行口工作的。在使用输入/输出函数 STDIO.H 库中的函数之前,应先对串行口进行初始化。例如,以 2400 波特率(时钟频率为 12MHz),初始化程序为:

```
SCON=0x52;
```

```
TMOD=0x20;
```

```
TH1=0xf3;
```

```
TR1=1;
```

当然也可以用其他的波特率。

在输入/输出函数 STDIO.H 中, 库中的所有其他的函数都依赖 `getkey()` 和 `putchar()` 函数, 如果希望支持其他 I/O 接口, 只须修改这两个函数。

函 数 名	函数原型	功 能
<code>getkey</code>	<code>extern char _getkey(void);</code>	从串口读入一个字符, 不显示
<code>getkey</code>	<code>extern char getkey(void);</code>	从串口读入一个字符, 并通过串口输出对应的字符
<code>putchar</code>	<code>extern char putchar(char c);</code>	从串口输出一个字符
<code>gets</code>	<code>extern char *gets(char *string,int len);</code>	从串口读入一个长度为 <code>len</code> 的字符串存入 <code>string</code> 指定的位置。输入以换行符结束。输入成功则返回传入的参数指针, 失败则返回 <code>NULL</code>
<code>ungetchar</code>	<code>extern char ungetchar(char c);</code>	将输入的字符送到输入缓冲区并将其值返回给调用者, 下次使用 <code>gets</code> 或 <code>getchar</code> 时可得到该字符, 但不能返回多个字符
<code>ungetkey</code>	<code>extern char ungetkey(char c);</code>	将输入的字符送到输入缓冲区并将其值返回给调用者, 下次使用 <code>_getkey</code> 时可得到该字符, 但不能返回多个字符
<code>printf</code>	<code>extern int printf(const char * fmtstr[,argument]...);</code>	以一定的格式通过 MCS-51 的串口输出数值或字符串, 返回实际输出的字符数
<code>sprintf</code>	<code>extern int(char * buffer,const char*fmtstr[,argument]);</code>	<code>sprintf</code> 与 <code>printf</code> 的功能相似, 但数据不是输出到串口, 而是通过一个指针 <code>buffer</code> , 送入可寻址的内存缓冲区, 并以 ASCII 码形式存放
<code>puts</code>	<code>extern int puts (const char * string);</code>	将字符串和换行符写入串行口, 错误时返回 <code>EOF</code> , 否则返回一个非负数
<code>scanf</code>	<code>extern int scanf(const char * fmtstr[,argument]...);</code>	以一定的格式通过 MCS-51 的串口读入数据或字符串, 存入指定的存储单元, 注意, 每个参数都必须是指针类型。scanf 返回输入的项数, 错误时返回 <code>EOF</code>
<code>sscanf</code>	<code>extern int sscanf(char *buffer,const char * fmtstr[,argument]);</code>	<code>sscanf</code> 与 <code>scanf</code> 功能相似, 但字符串的输入不是通过串口, 而是通过另一个以空结束的指针

4. 内部函数 INTRINS.H

函 数 名	函数原型	功 能
<code>_crol_</code> <code>_irol_</code> <code>_lrol_</code>	<code>unsigned char _crol_(unsigned char var,unsigned char n);</code> <code>unsigned int _irol_(unsigned int var,unsigned char n);</code> <code>unsigned long _lrol_(unsigned long var,unsigned char n);</code>	将变量 <code>var</code> 循环左移 <code>n</code> 位, 它们与 MCS-51 单片机的 <code>RL A</code> 指令相关。这 3 个函数的不同之处在于变量的类型与返回值的类型不一样
<code>_cror_</code> <code>_iror_</code> <code>_lror_</code>	<code>unsigned char _cror_(unsigned char var,unsigned char n);</code> <code>unsigned int _iror_(unsigned int var,unsigned char n);</code> <code>unsigned long _lror_(unsigned long var,unsigned char n);</code>	将变量 <code>var</code> 循环右移 <code>n</code> 位, 它们与 MCS-51 单片机的 <code>RR A</code> 指令相关。这 3 个函数不同之处在于变量的类型与返回值的类型不一样
<code>_nop_</code>	<code>void _nop_(void);</code>	产生一个 MCS-51 单片机的 <code>NOP</code> 指令
<code>_testbit_</code>	<code>bit _testbit_(bit b);</code>	产生一个 MCS-51 单片机的 <code>JBC</code> 指令。该函数对字节中的一位进行测试。如为 1 返回 1, 如为 0 返回 0。该函数只能对可寻址位进行测试

5. 标准函数 STD.LIB.H

函 数 名	函数原型	功 能
atof	float atof(void *string);	将字符串 string 转换成浮点数值并返回
atol	long atol(void *string);	将字符串 string 转换成整型数值并返回
atoi	int atoi(void *string);	将字符串 string 转换成整型数值并返回
calloc	void *calloc(unsigned int num,unsigned int len);	返回 n 个具有 len 长度的内存指针, 若无内存空间可用, 则返回 NULL。所分配的内存区域用 0 进行初始化
malloc	void *malloc(unsigned int size);	返回一个具有 size 长度的内存指针, 若无内存空间可用, 则返回 NULL。所分配的内存区域不进行初始化
realloc	void *realloc (void xdata *p,unsigned int size);	改变指针 p 所指向的内存单元的大小, 原内存单元的内容被复制到新的存储单元中, 如果该内存单元的区域较大, 多出的部分不做初始化。 realloc 函数返回指向新存储区的指针, 若无足够大的内存可用, 则返回 NULL
free	void free(void xdata *p);	释放指针 p 所指向的存储器区域, 如果返回值为 NULL, 则该函数无效, p 必须为以前用 callon、malloc 或 realloc 函数分配的存储器区域
init_mempool	void init_mempool(void *data *p,unsigned int size);	对被 callon、malloc 或 realloc 函数分配的存储器区域进行初始化。指针 p 指向存储器区域的首地址, size 表示存储区域的大小

6. 字符串函数 STRING.H

函 数 名	函数原型	功 能
memcpy	void *memcpy(void *dest,void *src,char val,int len);	复制字符串 src 中 len 个元素到字符串 dest 中。如果实际复制了 len 个字节则返回 NULL。复制过程在复制完字符 val 后停止, 此时返回指向 dest 中下一个元素的指针
memmove	void *memmove (void *dest,void *src,int len);	memmove 的工作方式与 memcpy 相同, 只是复制的区域可以交叠
memchr	void *memchr (void *buf,char c,int len);	顺序搜索字符串 buf 的头 len 个字符以找出字符 val, 成功后返回 buf 中指向 val 的指针, 失败时返回 NULL
memcmp	char memcmp(void *buf1,void *buf2,int len);	逐个字符比较串 buf1 和 buf2 的前 len 个字符, 相等时返回 0, 如 buf1 大于 buf2, 则返回一个正数; 如 buf1 小于 buf2, 则返回一个负数
memcpy	void *memcpy (void *dest,void *src,int len);	从 src 所指向的存储器单元复制 len 个字符到 dest 中, 返回指向 dest 中最后一个字符的指针
memset	void *memset (void *buf,char c,int len);	用 val 来填充指针 buf 中 len 个字符
strcat	char *strcat (char *dest,char *src);	将串 dest 复制到串 src 的尾部
strncat	char *strncat (char *dest,char *src,int len);	将串 dest 的 len 个字符复制到串 src 的尾部
strcmp	char strcmp (char *string1,char *string2);	比较串 string1 和串 string2, 相等则返回 0; string1>string2, 则返回一个正数; string1<string2, 则返回一个负数

(续表)

函 数 名	函数原型	功 能
strncmp	char strncmp(char *string1,char *string2,int len);	比较串 string1 与串 string2 的前 len 个字符, 返回值与 strcmp 相同
strcpy	char *strcpy (char *dest,char *src);	将串 src, 包括结束符, 复制到串 dest 中, 返回指向 dest 中第一个字符的指针
strncpy	char strncpy (char *dest,char *src,int len);	strcpy 与 strncpy 相似, 但它只复制 len 个字符。如果 src 的长度小于 len, 则 dest 串以 0 补齐到长度 len
strlen	int strlen (char *src);	返回串 src 中的字符个数, 包括结束符
Strchr strpos	char *strchr (const char *string,char c); int strpos (const char *string,char c);	strchr 搜索 string 串中第一个出现的字符 c, 如果找到则返回指向该字符的指针, 否则返回 NULL。被搜索的字符可以是串结束符, 此时返回值是指向串结束符的指针。strpos 的功能与 strchr 类似, 但返回的是字符 c 在串中出现的位置值或 -1, string 中首字符的位置值是 0
strlen	int strlen (char *src);	返回串 src 中的字符个数, 包括结束符
strrchr strrpos	char *strrchr (const char *string,char c); int strrpos (const char *string,char c);	strrchr 搜索 string 串中最后一个出现的字符 c, 如果找到则返回指向该字符的指针, 否则返回 NULL。被搜索的字符可以是串结束符, 此时返回值是指向串结束符的指针。strrpos 的功能与 strrchr 类似, 但返回的是字符 c 在串中最后一次出现的位置值或 -1
strspn strcspn strpbrk strrpbkr	int strspn(char *string,char *set); int strcspn(char *string,char * set); char *strpbrk (char *string,char *set); char *strrpbkr (char *string,char *set);	strspn 搜索 string 串中第一个不包括在 set 串中的字符, 返回值是 string 中包括在 set 里的字符个数。如果 string 中所有的字符都包括在 set 里面, 则返回 string 的长度 (不包括结束符), 如果 set 是空串则返回 0 strcspn 与 strspn 相似, 但它搜索的是 string 串中第一个包含在 set 里的字符。strpbrk 与 strspn 相似, 但返回指向搜索到的字符的指针, 而不是个数, 如果未搜索到, 则返回 NULL。strrpbkr 与 strpbrk 相似, 但它返回指向搜索到的字符的最后一个的字符指针

7. 数学函数 MATH.H

函 数 名	函数原型	功 能
abs cabs fabs labs	extern int abs(int i); extern char cabs(char i); extern float fabs(float i); extern long labs(long i);	计算并返回 i 的绝对值。这 4 个函数除了变量和返回值类型不同之外, 其他功能完全相同
exp log log10	extern float exp(float i); extern float log(float i); extern float log10(float i);	exp 返回以 e 为底的 i 的幂, log 返回 i 的自然对数(e = 2.718282), log10 返回以 10 为底的 i 的对数
sqrt	extern float sqrt(float i);	返回 i 的正平方根
rand srand	extern int rand(); extern void srand(int i);	rand 返回一个 0~32767 之间的伪随机数, srand 用来将随机数发生器初始化成一个已知的值, 对 rand 的相继调用将产生相同序列的随机数

(续表)


函 数 名	函数原型	功 能
cos sin tan	extern float cos(float i); extern float sin(float i); extern float tan(float i);	cos 返回 i 的余弦值, sin 返回 i 的正弦值, tan 返回 i 的正切值, 所有函数的变量范围都是 $-\pi/2 \sim +\pi/2$, 变量的值必须在 ± 65535 之间, 否则产生一个 NaN 错误
acos asin atan atan2	extern float acos(float i); extern float asin(float i); extern float atan(float i); extern float atan2(float i, float j);	acos 返回 i 的反余弦值, asin 返回 i 的反正弦值, atan 返回 i 的反正切值, 所有函数的值域都是 $-\pi/2 \sim +\pi/2$, atan2 返回 x/y 的反正切值, 其值域为 $-\pi \sim +\pi$
cosh sinh tanh	extern float cosh(float i); extern float sinh(float i); extern float tanh(float i);	cosh 返回 i 的双曲余弦值, sinh 返回 i 的双曲正弦值, tanh 返回 i 的双曲正切值

8. 绝对地址访问函数 ABSACC.H

函 数 名	函数原型	功 能
CBYTE DBYTE PBYTE XBYTE CWORD DWORD PWORD XWORD	#define CBYTE((unsigned char *)0x50000L) #define DBYTE((unsigned char *)0x40000L) #define PBYTE((unsigned char *)0x30000L) #define XBYTE((unsigned char *)0x20000L) #define CWORD((unsigned int *)0x50000L) #define DWORD((unsigned int *)0x50000L) #define PWORD((unsigned int *)0x50000L) #define XWORD((unsigned int *)0x50000L)	CBYTE 以字节形式对 CODE 区寻址; DBYTE 以字节形式对 DATA 区寻址; PBYTE 以字节形式对 PDATA 区寻址; XBYTE 以字节形式对 XDATA 区寻址; CWORD 以字形式对 CODE 区寻址; DWORD 以字形式对 DATA 区寻址; PWORD 以字形式对 PDATA 区寻址; XWORD 以字形式对 XDATA 区寻址。例如, XBYTE[0x0001]是以字节形式对片外 RAM 的 0001H 单元访问



参考文献

- [1] 王东锋. 单片机 C 语言应用 100 例[M]. 北京: 电子工业出版社, 2010
- [2] 杨旭方. 单片机控制与应用实训教程[M]. 北京: 电子工业出版社, 2010
- [3] 常敏. 51 单片机应用程序开发与实践[M]. 北京: 电子工业出版社, 2009
- [4] 谢维成. 单片机原理与应用及 C51 程序设计 (第 2 版) [M]. 北京: 清华大学出版社, 2009
- [5] 王元一. 单片机接口技术与应用 (C51 编程) [M]. 北京: 清华大学出版社, 2014
- [6] 孟祥莲. 单片机原理与应用—基于 Proteus 与 Keil C[M]. 哈尔滨工业大学出版社, 2010
- [7] 张欣. 单片机原理与 C51 程序设计基础教程[M]. 北京: 清华大学出版社, 2010
- [8] 张毅刚. 单片机原理及应用[M]. 北京: 高等教育出版社, 2004
- [9] 张欣. 单片机原理与 C51 程序设计教程[M]. 北京: 清华大学出版社, 2014 

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396；(010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路南口金家村 288 号华信大厦

电子工业出版社总编办公室

邮 编：100036